Começando com Linux

Comandos, serviços e administração

DANIEL ROMERO

Agradecimentos

Dedico este trabalho à minha esposa Mychelle, obrigado por compreender a minha ausência quando necessário e pelo apoio em todos os momentos. Aos meus pais pelo constante apoio e incentivo.

Agradeço a Casa do Código pela oportunidade de escrever esse livro, especialmente ao Paulo Silveira pelos ensinamentos e opiniões de muito valor sobre o conteúdo e organização do mesmo.

Agradecimentos especiais aos amigos Francieric Alves por mostrar o caminho das pedras e Fred Portela por ajudar a trilhar esse caminho. Agradeço também a todos os amigos que me ajudaram direta ou indiretamente na construção do livro.

Por fim, agradeço a Deus por mais esta oportunidade.

Sumário

Capítulo 1

Introdução

“Não há conhecimento que não tenha valor.”
– Edmund Burke

Raramente utilizamos um sistema operacional diretamente. O que usamos são programas, que utilizam recursos como arquivos, internet e memória, ambos providos pelo sistema operacional. Quando você usa o Internet Explorer e o Word, ambos pedem aos componentes internos do Windows, que é o verdadeiro sistema.

O Windows é um dos diversos sistemas operacionais. O Linux é outro deles.

O Linux ganhou muita popularidade e hoje encontra-se amplamente difundido nos servidores de grandes sistemas. Instalar bancos de dados e servidores web, além de gerenciá-los, é uma tarefa comum e relativamente fácil em sistemas Linux. Durante esse livro aprenderemos a trabalhar com arquivos, configurar servidores, compilar pacotes, criar scripts e realizar tarefas do dia a dia que um administrador costuma fazer.

A melhor forma de pensar em *Sistema Operacional* é imaginar um conjunto de vários programas unidos, aguardando serem usados. Com essa definição em mente

podemos dizer que distribuições *Linux* como Slackware, Debian, Ubuntu, Red Hat e Fedora são um aglomerado de programas e configurações específicas. São sabores diferentes do Linux. Há vantagens e desvantagens em cada uma delas.

Há também o Android, que adaptou o kernel do Linux para criar seu próprio sistema operacional focado em dispositivos móveis.

1.1 Para quem é este livro

Este livro tem o objetivo de apresentar uma introdução ao *Linux* de forma bastante prática e com uso de muitos exemplos. É recomendado para iniciantes e pode ser um bom guia de consultas para usuários com mais experiência.

Existe uma lista de discussões aqui:

http://lista.infoslack.com

Sinta-se à vontade para mandar dúvidas sobre o livro. Além disso, sugestões, críticas e correções serão bem vindas.

1.2 O que é Linux?

Linux é o *kernel*, o componente central, o coração do sistema. Ele é responsável por conectar os programas ao hardware. Pense no *Linux* como parte central de um *Sistema Operacional*.

Apenas com o Kernel não teríamos muitas opções. Precisamos de programas, configurações, interface gráfica e drivers para tirar um proveito real desse sistema operacional.

Para isso, utilizaremos a distribuição de Linux conhecida como Ubuntu.

http://www.ubuntu.com/

1.3 Por que o Ubuntu?

Pensando em iniciantes, escolhi o Ubuntu para usar durante todo o livro, pois é um *Sistema Operacional* fácil de usar, instalar e configurar, além de possuir uma vasta documentação.

Nós vamos, nesse capítulo, realizar a instalação do Ubuntu. Mas há outras opções, no caso de você preferir só testá-lo.

Isso é possível pois as versões para download do Ubuntu são *Live CD* e possibilita que você execute todo o sistema operacional sem instalar nada, tudo será executado diretamente na memória RAM.

Outra opção é fazer uso de máquinas virtuais, como é o caso do VirtualBox https://www.virtualbox.org/ e VMware http://www.vmware.com/br/, e ter um sistema operacional virtualizado dentro de outro.

Existe outra solução mais elegante que faz uso tanto do VirtualBox quanto do VMware no seu background, o Vagrant http://www.vagrantup.com/.

1.4 Instalação do Ubuntu

O Ubuntu foi projetado para ter uma instalação rápida e muito fácil. Durante este processo ele pode baixar da internet algumas atualizações do sistema e pacotes de linguagem para o seu idioma. Dependendo da velocidade da sua conexão o tempo da instalação pode demorar um pouco.

Apresentarei a forma padrão de instalação do Ubuntu sem conexão com a internet, mas você pode optar por testá-lo antes de instalar executando-o direto de um CD/DVD ou pendrive.

A seguir temos a tela de boas vindas do instalador do Ubuntu:

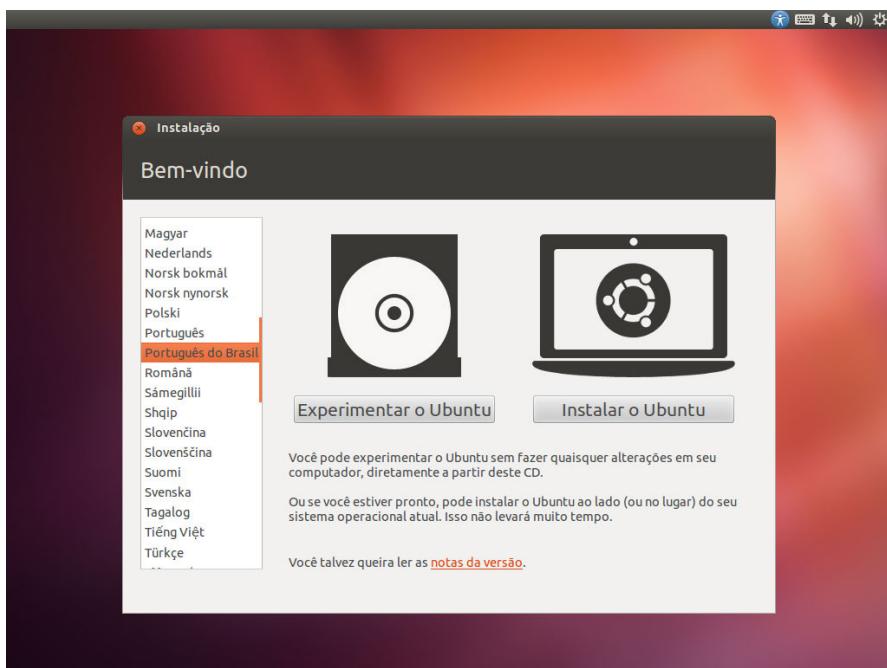



Figura 1.1: Tela de boas vindas

Nessa tela temos a opção de escolha do idioma — no caso selecionei *Português do Brasil*. Clique em *Instalar o Ubuntu* para continuarmos o processo de instalação.

Agora o Ubuntu irá verificar a quantidade de espaço disponível em disco. Ele precisa de, no mínimo, 4.4GB para prosseguir a instalação.

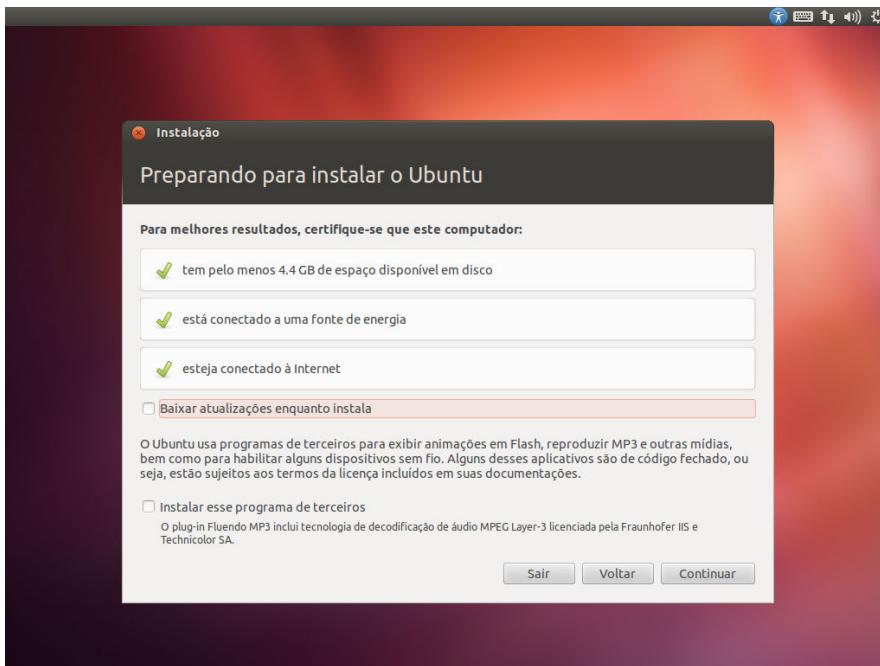



Figura 1.2: Preparando para instalar o Ubuntu

Note que não marquei as opções de baixar atualizações enquanto instala e instalar programas de terceiros pode fazer a instalação demorar um pouco além do previsto.

Vamos prosseguir escolhendo agora o tipo de instalação.

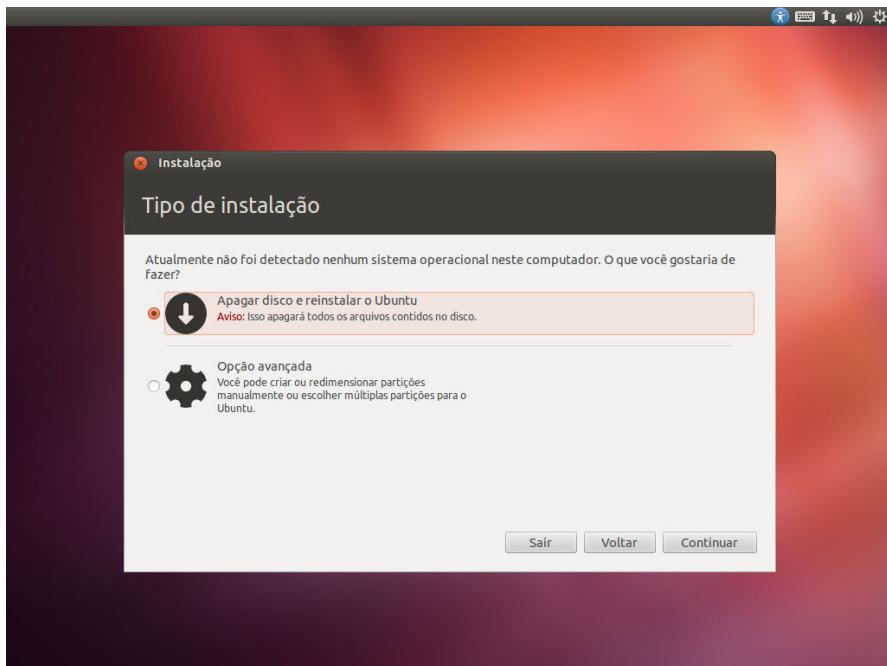



Figura 1.3: Seleção de tipo de instalação

Observe a mensagem que diz **Atualmente não foi detectado nenhum sistema operacional neste computador.**. Como não tenho nada instalado marquei a opção de apagar o disco e instalar o Ubuntu.

Outras opções são oferecidas quando já possuímos um sistema instalado, como instalar o Ubuntu ao lado do Windows ou substituir o Windows completamente pelo Ubuntu.

Na sequência, o instalador informa que vai apagar o disco inteiro e prosseguir:

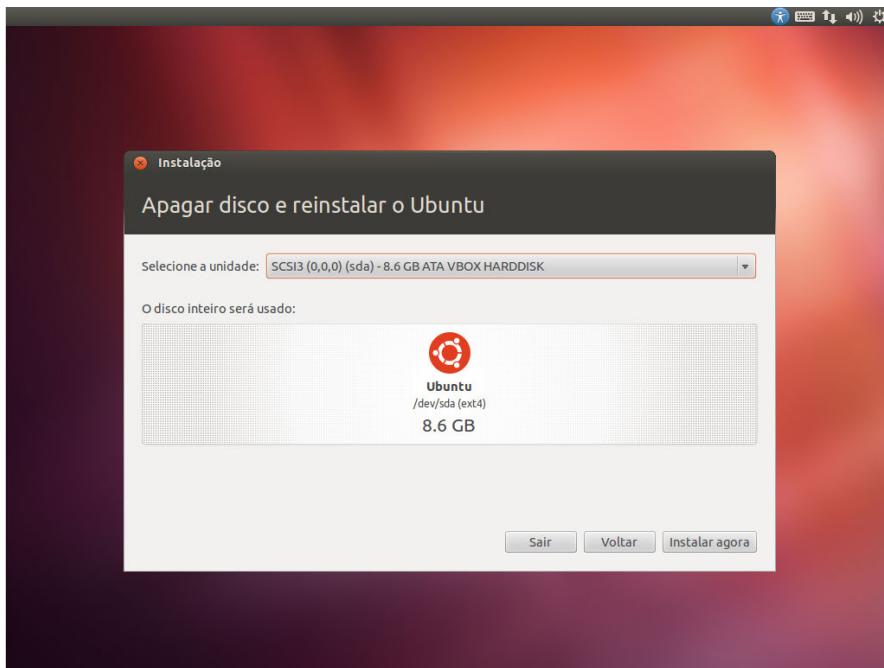



Figura 1.4: Disco que será usado

Durante o processo, o instalador do Ubuntu solicita a sua localização para seleção de fuso horário.

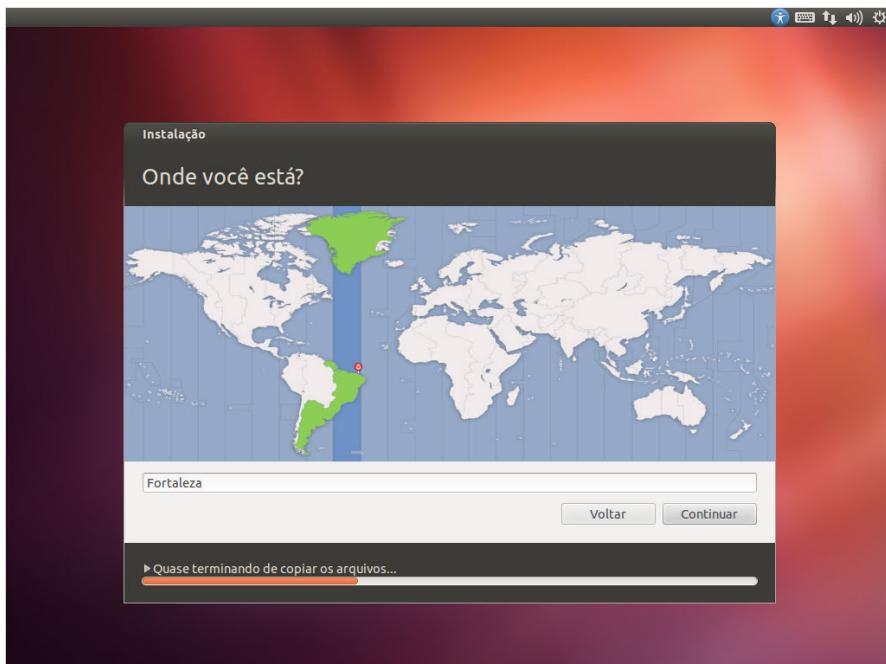



Figura 1.5: Seleção de fuso horário

Em seguida, devemos escolher o layout do nosso teclado. Por padrão, o instalador do Ubuntu tenta reconhecer o tipo de teclado automaticamente.

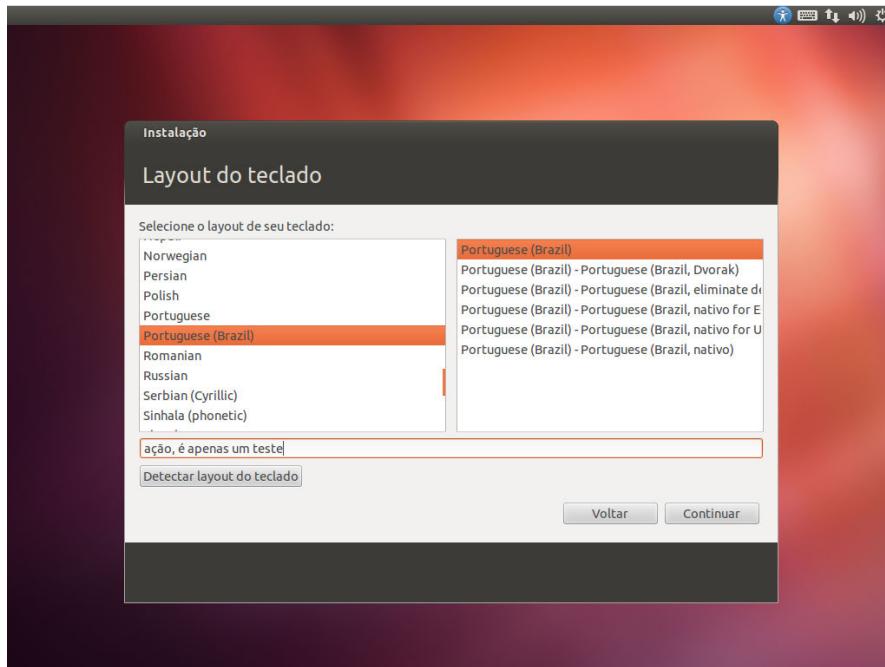



Figura 1.6: Escolha do layout do teclado

Antes de concluir a instalação, o sistema pede algumas informações: o seu nome, nome do computador, um nome de usuário e senha.

Existe a opção `Criptografar minha pasta pessoal`, que insere uma proteção complementar aos seus arquivos armazenados no disco.

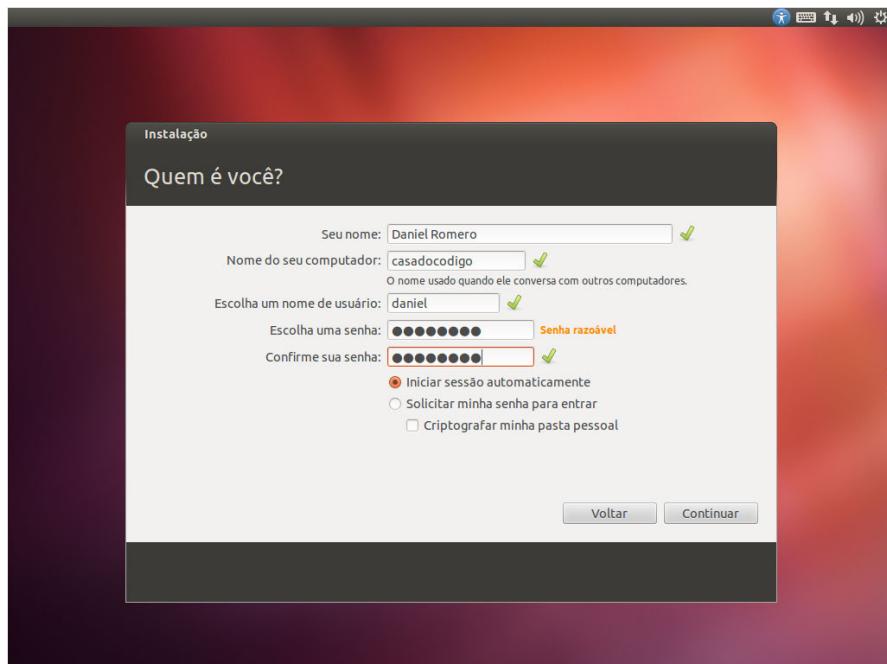



Figura 1.7: Informando os dados do usuário

O processo de finalização pode levar alguns minutos e o instalador oferece nesse tempo um guia ilustrado sobre o Ubuntu.

Figura 1.8: Processo de instalação

Ao finalizar, teremos a mensagem informando que é necessário reiniciar o computador. Clique em `Reiniciar agora`.

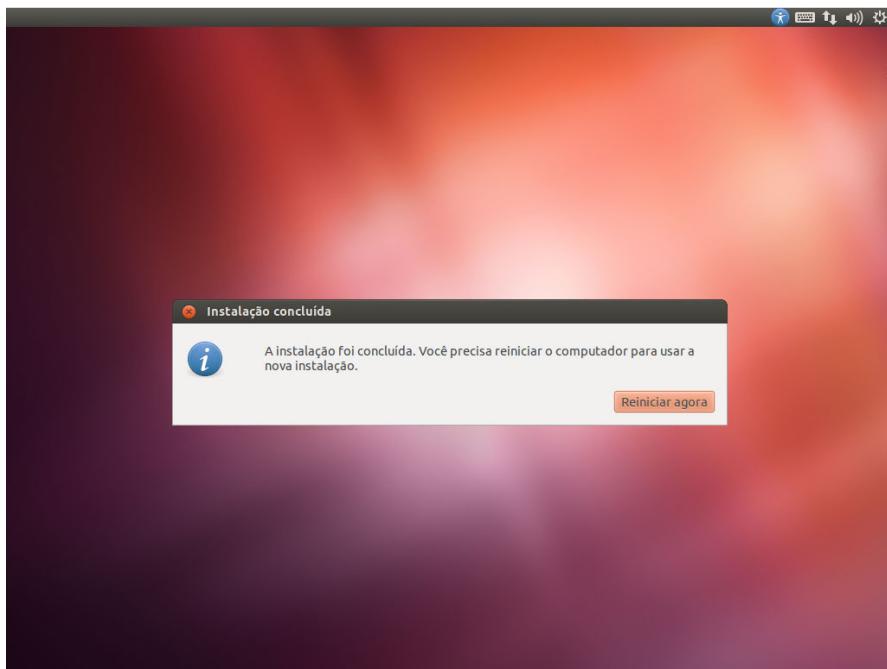



Figura 1.9: Finalizando a instalação

O sistema irá reiniciar o computador e finalmente teremos o Ubuntu pronto para uso. Daqui pra frente tudo o que veremos no livro será executado no terminal de comandos.

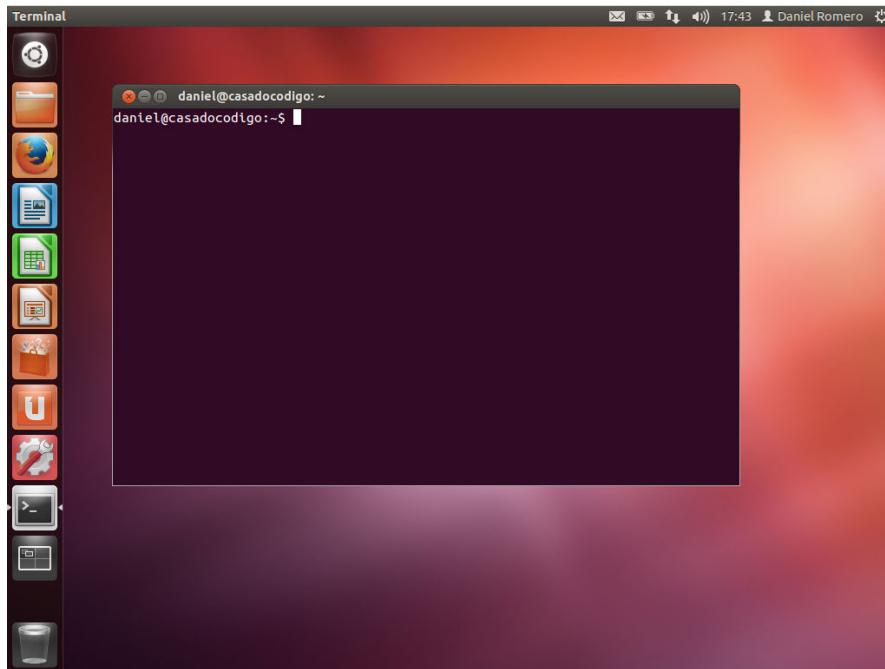



Figura 1.10: Ubuntu desktop

1.5 Navegando no novo sistema operacional

Vamos explorar um pouco o novo sistema operacional, abrindo alguns programas:

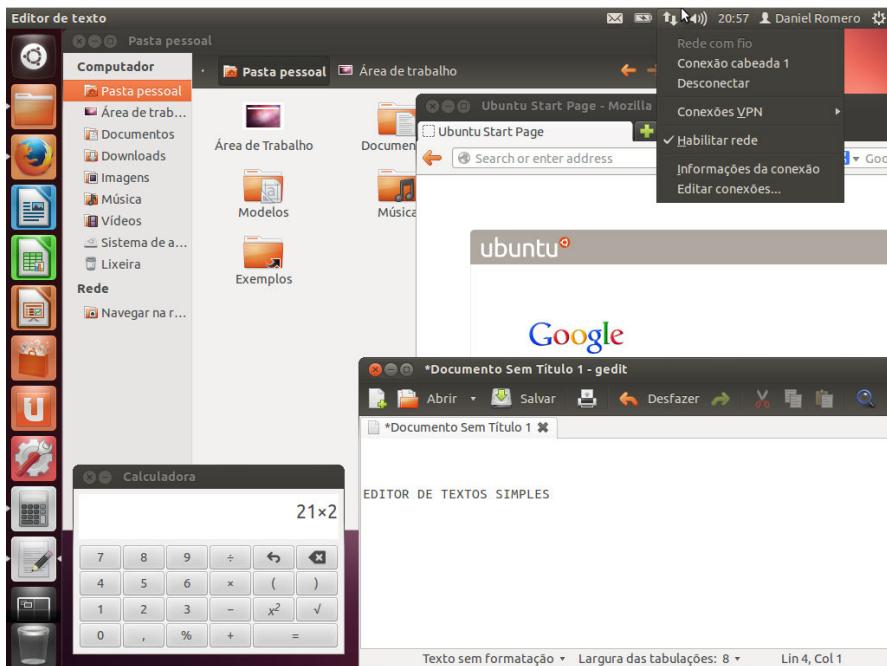



Figura 1.11: Alguns aplicativos abertos

Experimente o novo ambiente, abra alguns aplicativos como no exemplo anterior.

Observe como é familiar com outros sistemas operacionais, editor de texto , a calculadora, o gerenciador de pastas e arquivos, o browser, configurações de rede etc.

Por padrão, o Ubuntu já instala uma suíte de aplicativos de escritório gratuita: o LibreOffice, que possui compatibilidade com os arquivos formatados no Microsoft Office. Confira:

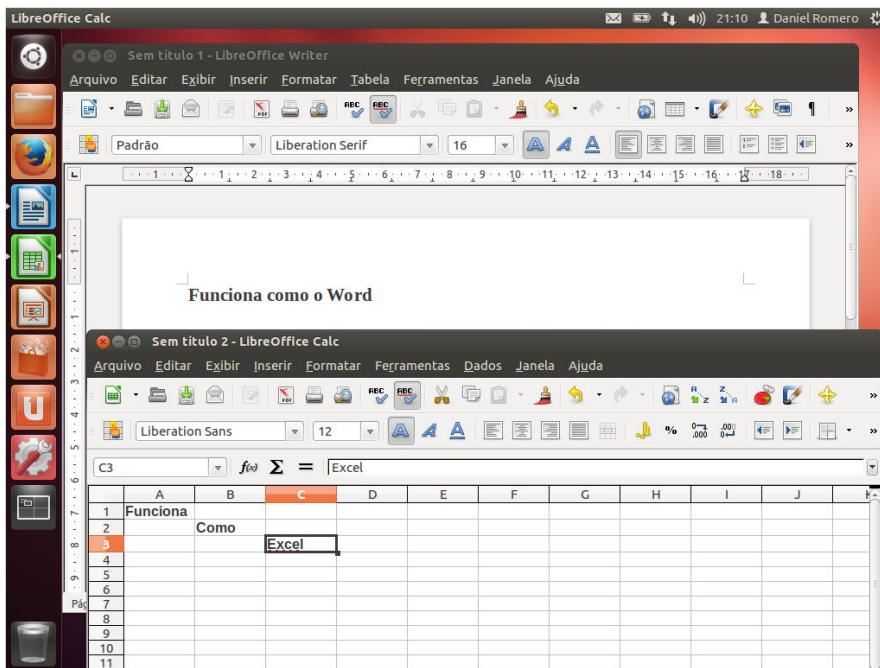



Figura 1.12: LibreOffice, equivalentes ao Word e Excel

Se estiver procurando um semelhante ao `notepad` do Windows, o Ubuntu possui o `Gedit`:

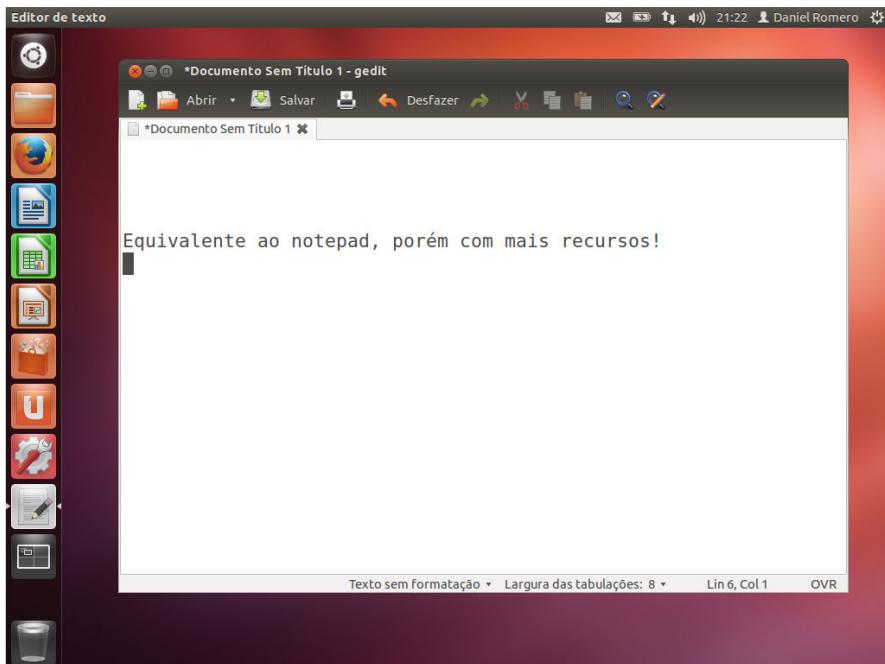



Figura 1.13: Gedit, equivalente ao notepad do Windows

Com ele você pode criar e alterar arquivos. Nesse livro veremos as formas mais comuns de fazer isso, que será através de editores menos visuais, porém poderosos.

Para alterar as configurações de sistema e personalizar de acordo com suas preferências, por exemplo, o visual, configurações de rede, teclado, mouse, data e muito mais, vá em `Configurações do sistema`:

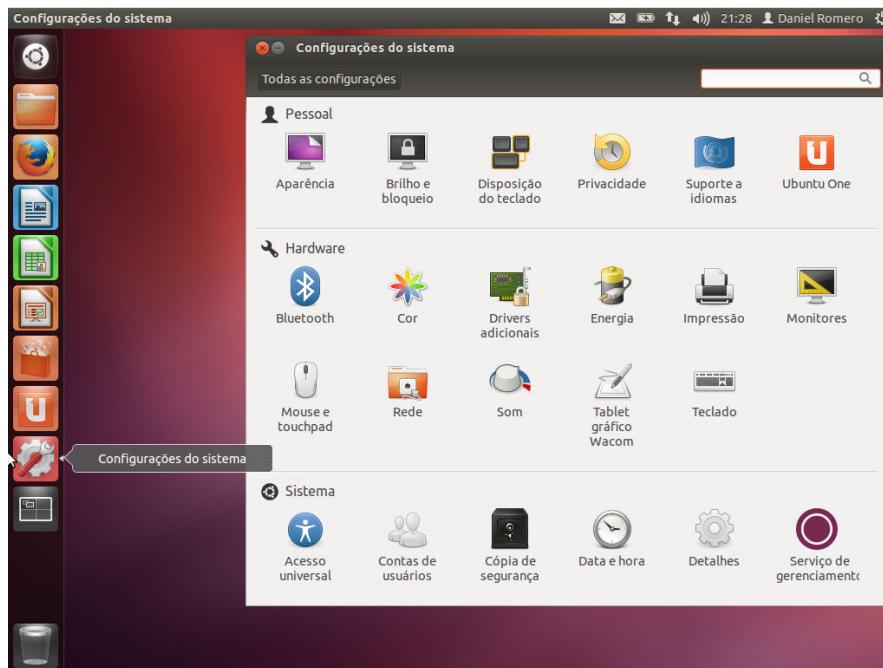



Figura 1.14: Configurações do sistema

Você pode notar que usamos o Ubuntu como um sistema operacional tradicional, assim como nossos amigos costumam usar o Windows.

Apesar dessa facilidade, focaremos o livro no uso para desenvolvedores e administradores de sistemas. Sem dúvida, a ferramenta principal é o *shell* do Linux, que veremos agora.

Capítulo 2

Mão na massa: conhecendo o Shell

Neste capítulo veremos um pouco sobre o *Shell* de forma prática, alguns comandos de navegação, informação, busca e manipulação de arquivos e diretórios.

2.1 O que é o Shell?

O `shell` é a interface de acesso ao sistema operacional, onde é possível interagir com o sistema por meio de comandos digitados do teclado. Ele pode ser acessado pelo modo gráfico e diretamente em modo texto. Veremos detalhes sobre isso no decorrer do livro.

2.2 Primeiro contato com o Shell

Após instalar o Ubuntu e abrir o terminal de comandos, você se depara com a seguinte imagem:

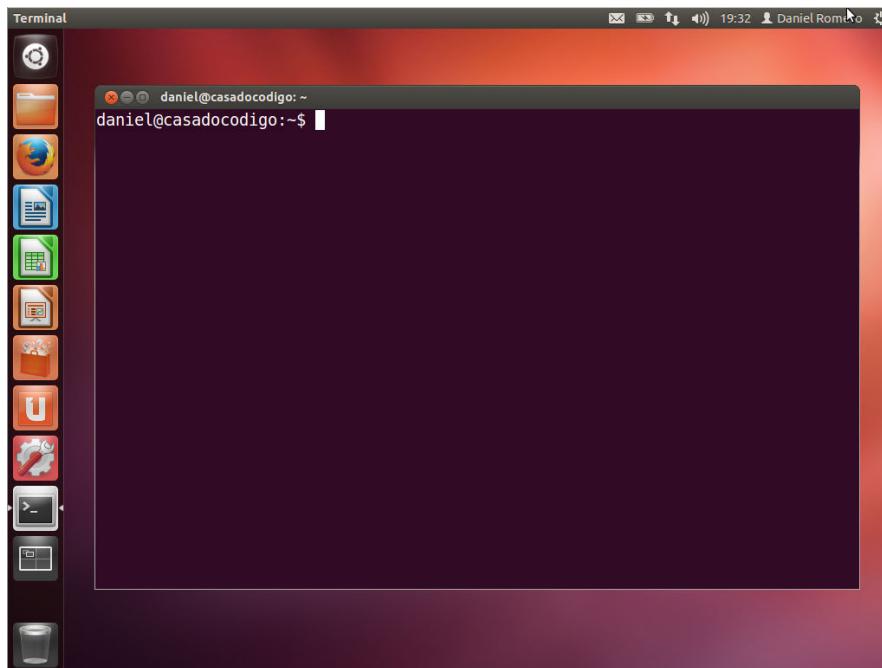



Figura 2.1: Primeiro contato com o Shell

Mas o que significa isso?

- `ubuntu` é o nome do usuário;
- `@servidor` é o nome do host;
- `~` é a abreviação para a pasta do usuário;
- `$` indica que o usuário não possui poderes de administrador.

Agora que entendemos o significado do `prompt` que aparece, vamos testar alguns comandos e ver o que acontece.

```
daniel@casadocodigo:~$ pwd
/home/daniel
daniel@casadocodigo:~$ whoami
daniel
daniel@casadocodigo:~$ date
Thu Sep 26 21:30:07 UTC 2013
daniel@casadocodigo:~$
```

Ao executar o comando `pwd`, o retorno foi exatamente o diretório que estamos atualmente chamando de *current*.

O comando `whoami` exibe o nome do usuário atual que estamos usando e o `date` retorna a data atual.

Não se preocupe ainda sobre os poderes de administrador `sudo`, veremos isso mais adiante.

Sempre que quisermos digitar algum comando que já tenha sido executado antes, podemos fazer uso da tecla `direcional para cima` ou `direcional para baixo`. Isso possibilita navegarmos por uma lista de comandos que já foram executados.

Existe a opção de vermos essa lista completa, graças ao `history`, que cria um histórico memorizando tudo o que já digitamos no `shell`:

```
daniel@casadocodigo:~$ history
    1  pwd
    2  whoami
    3  date
    4  clear
    5  history
daniel@casadocodigo:~$
```

Outra recomendação para um uso mais produtivo do `shell` é utilizar a tecla `tab`, que possui a função de autocompletar.

2.3 Trabalhando com diretórios

Para conhecer mais sobre o Shell, vamos navegar por alguns diretórios e executar alguns comandos básicos para trabalhar com arquivos. À medida que formos navegando na estrutura *Linux* entenderemos como ele é organizado.

Antes de continuar, observe a estrutura de diretórios no ambiente gráfico do Ubuntu:

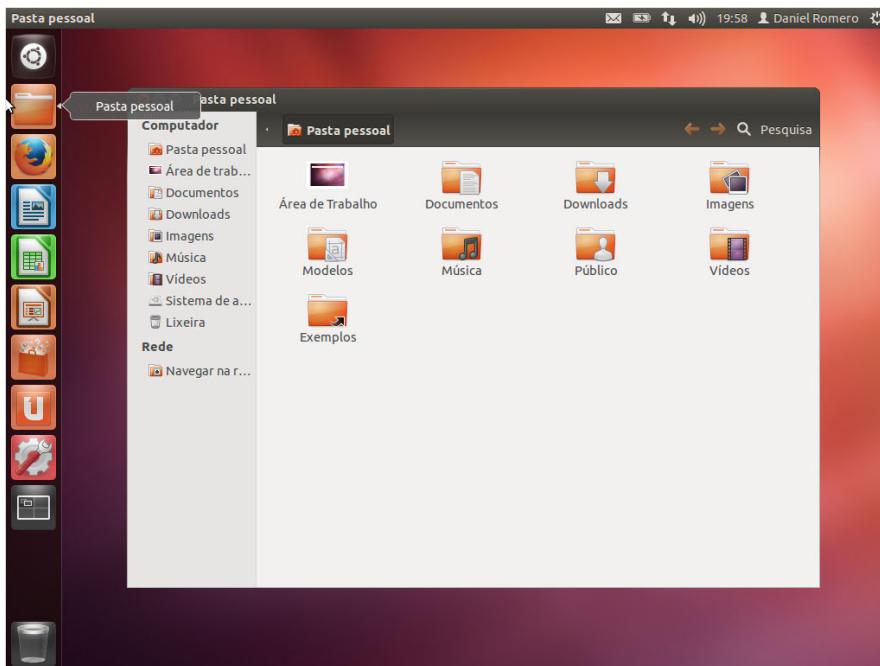



Figura 2.2: Listagem de diretórios na pasta home

Em alguns sistemas operacionais, é utilizado o termo *pasta* (*folder*) em vez de *diretório* (*directory*).

```
daniel@casadocodigo:~$ cd /
daniel@casadocodigo:/$
```

Ao executar o comando `cd /` estamos informando ao Shell que queremos navegar até o diretório `/`, mais conhecido como raiz (ou diretório *root*). A instrução `cd` é o que permite a navegação entre diretórios.

Já o comando `ls` lista os arquivos e diretórios, neste caso no diretório raiz, que é onde estamos. Faça o teste!

```
daniel@casadocodigo:/$ ls
bin    home         media  Repos  selinux  usr
boot   initrd.img   mnt    root   srv      vagrant
dev    lib          opt    run    sys      var
etc    lost+found   proc   sbin   tmp      vmlinuz
daniel@casadocodigo:/$
```

A mesma listagem no ambiente gráfico:

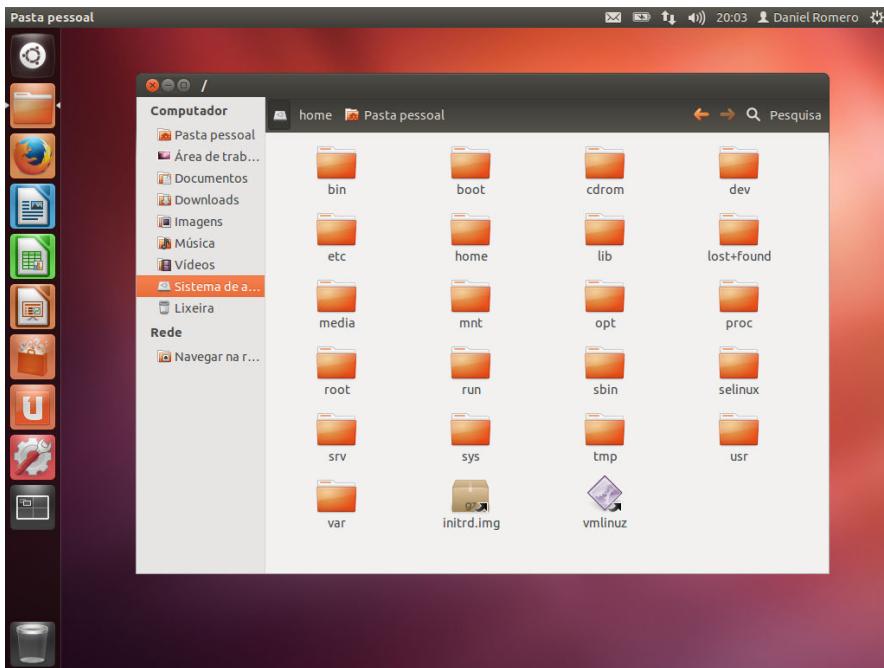



Figura 2.3: Listagem do diretório raiz

Não se preocupe, veremos cada um desses diretórios da listagem de forma detalhada em um capítulo apropriado.

Sua instalação pode listar arquivos um pouco diferentes, não tem problema algum. Eles variam de acordo com instalação.

Perceba que algumas vezes, após digitarmos comandos ou listarmos conteúdo de diretórios, a tela do terminal fica repleta de palavras. Para fazer a limpeza da tela usamos o comando `clear`. Teste o comando e veja o resultado. O atalho `Ctrl + l` tem o mesmo efeito.

Vamos retornar ao nosso diretório de usuário que fica em `/home/ubuntu`, que é conhecido por `home`. Para isso vamos usar o `cd`.

Para retornar, temos algumas opções para combinar com o comando `cd`:

- ~ aponta direto para o diretório *home* do usuário (ou *pasta do usuário*);
- `/home/ubuntu` é o caminho absoluto;

- – último diretório onde esteve.

No exemplo a seguir usaremos o `cd -` para retornar ao diretório `home`:

```
daniel@casadocodigo:/$ ls
bin   home        media  Repos  selinux  usr
boot  initrd.img  mnt    root   srv      vagrant
dev   lib         opt    run    sys      var
etc   lost+found  proc   sbin   tmp      vmlinuz
daniel@casadocodigo:/$ cd -
/home/daniel
daniel@casadocodigo:~$
```

Teste as outras combinações e se ficar perdido retorne imediatamente ao diretório home, executando `cd ~`.

Agora que voltamos ao nosso diretório home, vamos criar um novo diretório usando o comando `mkdir`. Chamaremos de `aplicativos`:

```
daniel@casadocodigo:~$ pwd
/home/daniel
daniel@casadocodigo:~$ mkdir aplicativos
daniel@casadocodigo:~$ ls
aplicativos
daniel@casadocodigo:~$
```

Agora que o diretório foi criado, vamos entrar e sair dele testando mais duas novas formas de navegação usando o `cd`, seguido do nome do diretório:

```
daniel@casadocodigo:~$ ls
aplicativos
daniel@casadocodigo:~$ cd aplicativos/
daniel@casadocodigo:~/aplicativos$ cd ..
daniel@casadocodigo:~$
```

Observe que não passamos o **caminho absoluto** para o `cd`, apenas o **caminho relativo**, pois o diretório em que queremos entrar é **relativo** ao que estamos atualmente.

Alternativamente, você poderia sim ter utilizado o caminho absoluto, colocando todo o diretório, desde a raiz, no comando, fazendo `cd /home/daniel/aplicativos`.

Vale lembrar que o diretório `daniel` depende do nome de usuário que você criou durante a instalação do Ubuntu.

Para voltar, usamos o `cd ..` que significa *volte para o diretório anterior*.

2.4 Trabalhando com arquivos

Vamos criar um arquivo no diretório `home` do nosso usuário e, em seguida, movê-lo para dentro do diretório que foi criado, o `aplicativos`:

```
daniel@casadocodigo:~$ ls
aplicativos
daniel@casadocodigo:~$ touch meu_arquivo
daniel@casadocodigo:~$ ls
aplicativos  meu_arquivo
daniel@casadocodigo:~$ mv meu_arquivo aplicativos/
daniel@casadocodigo:~$ ls
aplicativos
daniel@casadocodigo:~$ ls aplicativos/
meu_arquivo
daniel@casadocodigo:~$
```

Utilizando o comando `touch`, criamos um novo arquivo chamado `meu_arquivo`. Em seguida usamos o comando `mv`, passando como parâmetro o nome do arquivo e o destino para mover o arquivo que criamos. Note que ao movermos o arquivo de lugar ele não fica mais disponível onde estava antes.

No caso anterior o `meu_arquivo` foi movido para o diretório `aplicativos`, mas e se quisermos ter uma cópia desse arquivo na pasta `home` do nosso usuário?

Para isso, existe o comando `cp`, que faz uma cópia do arquivo mantendo o original. Ele funciona de forma parecida com o `mv`:

```
daniel@casadocodigo:~$ ls
aplicativos
daniel@casadocodigo:~$ cp aplicativos/meu_arquivo .
daniel@casadocodigo:~$ ls
aplicativos  meu_arquivo
daniel@casadocodigo:~$ ls aplicativos/
meu_arquivo
daniel@casadocodigo:~$
```

Observe a legibilidade: estamos falando para o shell copiar o arquivo `meu_arquivo` que se encontra em `aplicativos` para o diretório atual, e tudo foi feito sem sair da pasta `home`. O mesmo poderia ser feito de outra forma: `cp aplicativos/meu_arquivo /home/daniel`, passando o caminho absoluto do diretório destino para a cópia.

Antes de prosseguir vamos ver um pouco sobre arquivos ocultos. Um arquivo oculto é identificado por um "." no início do nome, por exemplo (.profile). Arquivos ocultos não aparecem em listagens normais de diretórios, para isso usamos um parâmetro no comando ls, ls -a ou, no formato mais detalhado, ls -la. No Ubuntu existe um atalho para esse comando, o ll.

Vamos listar os arquivos ocultos da pasta *home*:

```
daniel@casadocodigo:~$ ls -a
.   aplicativos    .bash_logout  meu_arquivo
..  .bash_history  .bashrc       .profile
daniel@casadocodigo:~$
```

Agora vamos listar os arquivos ocultos de forma mais detalhada:

```
daniel@casadocodigo:~$ ls -la
total 28
drwxr-xr-x 3 daniel daniel 4096 Sep 27 01:29 .
drwxr-xr-x 4 root   root   4096 Sep 11 02:35 ..
drwxrwxr-x 2 daniel daniel 4096 Sep 27 01:28 aplicativos
-rw------- 1 daniel daniel  697 Sep 26 21:58 .bash_history
-rw-r--r-- 1 daniel daniel  220 Sep 11 02:35 .bash_logout
-rw-r--r-- 1 daniel daniel 3486 Sep 11 02:35 .bashrc
-rw-rw-r-- 1 daniel daniel    0 Sep 27 01:29 meu_arquivo
-rw-r--r-- 1 daniel daniel  675 Sep 11 02:35 .profile
daniel@casadocodigo:~$
```

Não se preocupe com todas essas informações exibidas na listagem detalhada, veremos o significado de cada uma em capítulos posteriores. Agora continuaremos a trabalhar com arquivos.

O comando mv também serve para renomear um arquivo. Vamos renomear o arquivo que criamos meu_arquivo para meu_arquivo.txt, adicionando a extensão .txt:

```
daniel@casadocodigo:~$ ls
aplicativos  meu_arquivo
daniel@casadocodigo:~$ mv meu_arquivo meu_arquivo.txt
daniel@casadocodigo:~$ ls
aplicativos  meu_arquivo.txt
daniel@casadocodigo:~$
```

Nosso arquivo atualmente está vazio. Vamos escrever algum texto nele e conhecer mais um comando, o `echo`. O `echo` é utilizado para exibir mensagens na tela ou em um arquivo:

```
daniel@casadocodigo:~$ echo "Exibindo mensagem na tela"
Exibindo mensagem na tela
daniel@casadocodigo:~$
```

Agora vamos escrever algo no nosso arquivo com o `echo`, mas para isso é necessário fazer uso do parâmetro `>`, confira:

```
daniel@casadocodigo:~$ echo Escrevendo no arquivo > meu_arquivo.txt
```

Note como é legível, a instrução pode ser lida da seguinte forma: *escreva o texto "Escrevendo no arquivo" em meu_arquivo.txt.*

Agora podemos verificar o texto que foi inserido no nosso arquivo, com o comando `cat`:

```
daniel@casadocodigo:~$ ls
aplicativos  meu_arquivo.txt
daniel@casadocodigo:~$ cat meu_arquivo.txt
Escrevendo no arquivo
daniel@casadocodigo:~$
```

O `cat` faz parte de uma coleção de comandos para manipular arquivos, veremos mais detalhes sobre ele no próximo capítulo.

É interessante saber informações sobre o tipo de arquivo, para isso existe o comando `file`:

```
daniel@casadocodigo:~$ ls
aplicativos  meu_arquivo.txt
daniel@casadocodigo:~$ file meu_arquivo.txt
meu_arquivo.txt: ASCII text
daniel@casadocodigo:~$ file aplicativos/
aplicativos/: directory
daniel@casadocodigo:~$
```

Com o `file` podemos ver que o nosso arquivo é do tipo *text*, nota-se também que ao executar o `file` em um diretório, ele informa o tipo *directory*. Isso acontece pois no Linux tudo é considerado um arquivo.

Agora que já sabemos criar arquivos, e manipular de forma básica, hora de aprender a remover. Usando o `rm` podemos deletar arquivos e diretórios:

```
daniel@casadocodigo:~$ ls
aplicativos  meu_arquivo
daniel@casadocodigo:~$ rm meu_arquivo
daniel@casadocodigo:~$ ls
aplicativos
daniel@casadocodigo:~$ rm aplicativos/
rm: cannot remove `aplicativos/': Is a directory'`
daniel@casadocodigo:~$
```

Note que conseguimos deletar o arquivo `meu_arquivo`, porém ao tentar remover o diretório `aplicativos` recebemos uma mensagem de erro informando que não foi possível, pois trata-se de um diretório.

O comando `rmdir` é responsável por remover diretórios, mas somente diretórios vazios. No nosso caso `aplicativos` contém uma cópia de `meu_arquivo`, então o que fazer?

Usando o `rm -r` vamos conseguir remover o diretório, o `-r` é uma opção do comando `rm` que faz ele remover diretórios que tenham conteúdo de forma recursiva:

```
daniel@casadocodigo:~$ ls
aplicativos
daniel@casadocodigo:~$ rm -r aplicativos/
daniel@casadocodigo:~$ ls
daniel@casadocodigo:~$
```

Obviamente, tenha muito cuidado ao utilizá-lo!

2.5 Pedindo ajuda

Sempre que precisarmos de ajuda no Linux, podemos usar o `help` ou podemos incluir o parâmetro `--help`, que é suportado na maioria dos comandos. Vejamos alguns exemplos:

```
daniel@casadocodigo:~$ help cd
cd: cd [-L|[-P [-e]]] [dir]
    Change the shell working directory.
```

Ao usar o comando `help`, estou pedindo ajuda ao **shell** para que me informe mais detalhes sobre como usar um determinado comando, neste caso o `cd`.

Como falei, outra forma de ajuda é o parâmetro `--help`, que pode ser complemento para a maioria dos comandos. Veja como um exemplo de uso:

```
daniel@casadocodigo:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
```

No Linux a documentação padrão é chamada de `man pages`, ela contém ajuda para todos os comandos padrões. Vamos ao teste, execute o comando `man ls` e veja o que ele retorna:

```
daniel@casadocodigo:~$ man ls
```

```
LS(1)                         User Commands                        LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about the FILEs (the current directory by
       default).  Sort entries alphabetically if none of  -cftuvSUX
       nor --sort is specified.

       Mandatory  arguments to long options are mandatory for short
       options too.
 Manual page ls(1) line 1 (press h for help or q to quit)
```

Figura 2.4: Exibindo a documentação de ajuda de um comando

O `man` apresenta todos os tópicos do manual linux, de forma semelhante a um manual impresso e está presente em qualquer distribuição linux. O ambiente apresentado pelo `man` aceita comandos para navegação e busca na documentação:

- `h` mostra a ajuda do comando man, veja para mais detalhes;
- `q` sair do man;
- `/` procura por um texto na documentação apresentada;
- `f` avança uma tela;
- `b` volta uma tela.

Teste cada um dos comandos e verifique os resultados, tecle `h` para ver todas as opções do `man`:

```
                    SUMMARY OF LESS COMMANDS

     Commands marked with * may be preceded by a number, N.
     Notes in parentheses indicate the behavior if N is given.

 h  H                 Display this help.
 q  :q  Q  :Q  ZZ     Exit.
 ---------------------------------------------------------------------------

                            MOVING

 e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
 y  ^Y  k  ^K  ^P  *  Backward one line   (or N lines).
 f  ^F  ^V  SPACE  *  Forward  one window (or N lines).
 b  ^B  ESC-v      *  Backward one window (or N lines).
 z                 *  Forward  one window (and set window to N).
 w                 *  Backward one window (and set window to N).
 ESC-SPACE         *  Forward  one window, but don't stop at end-of-file.
HELP -- Press RETURN for more, or q when done
```

Figura 2.5: Verificando as opções do man

Tecle `q` para voltar a documentação que estava aberta e faça uma busca por algum texto na documentação, use o comando `/`:

```
LS(1)                        User Commands                       LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the  FILEs  (the  current  directory  by
       default).  Sort entries alphabetically if  none  of  -cftuvSUX  nor
       --sort is specified.

       Mandatory arguments to long options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

/human
```

Figura 2.6: Busca por texto na documentação

Podemos observar que o texto que buscamos foi selecionado destacando a cor de fundo:

```
       -h, --human-readable
              with -l, print sizes in human readable format (e.g., 1K 234M
              2G)

       --si   likewise, but use powers of 1000 not 1024

       -H, --dereference-command-line
              follow symbolic links listed on the command line

       --dereference-command-line-symlink-to-dir
              follow  each  command  line  symbolic  link that points to a
              directory

       --hide=PATTERN
              do not list implied entries matching shell PATTERN (overrid-
              den by -a or -A)

       --indicator-style=WORD
 Manual page ls(1) line 81 (press h for help or q to quit)
```

Figura 2.7: Resultado da busca por texto

Para sair, tecle `q`.

Claro, a documentação é toda em inglês e este tipo de tela pode não ser muito amigável. Na internet, muitas vezes é possível encontrar exemplos e tutoriais mais simples sobre cada comando, até mesmo em português!

Mas é importante você encarar o `man` e sua navegação estranha. Você verá que, no Linux, muitos comandos trabalham de forma similar.

Obtendo a descrição de comandos

É interessante saber que podemos verificar uma descrição simples de um comando, para isso podemos usar o `whatis`:

```
daniel@casadocodigo:~$ whatis ls
ls (1)               - list directory contents
daniel@casadocodigo:~$ whatis man
man (1)              - an interface to the on-line reference manuals
man (7)              - macros to format man pages
daniel@casadocodigo:~$
```

2.6 Localizando arquivos no sistema

Veremos agora um pouco sobre o comando `find`, ele serve para procurar arquivos pelo nome e outras características como data de modificação, tamanho, usuário criador do arquivo etc.

A sintaxe do `find` recebe alguns parâmetros:

```
find [caminho] expressão [ação]
```

Indicamos o *caminho* a partir do qual ele irá procurar os arquivos, a *expressão* na qual podemos definir os critérios de busca e a *ação* com a qual informamos o que deve ser feito com os arquivos que atenderem aos critérios da busca.

Alguns dos critérios de busca definidos em *expressão*:

- `-name` procura arquivos pelo nome;
- `-user` procura arquivos pelo nome do usuário dono do arquivo;
- `-atime` procura arquivos que foram acessados há mais de x dias, onde x é o número de dias.

Para saber mais sobre o `find` leia sempre a documentação, use o `man`!

Vamos praticar um pouco buscando arquivos com o `find`, para isso vamos começar listando na tela todos os arquivos ocultos que contenham a palavra `bash`:

```
daniel@casadocodigo:~$ find . -name \*.bash*
./.bash_history
./.bashrc
./.bash_logout
daniel@casadocodigo:~$
```

Perceba que informamos o caminho `.` (que significa *aqui*), a expressão `-name \*.bash*`, que quer dizer "busque pelo nome todos os arquivos que contenham a palavra bash no corpo do seu nome", e a ação `-print` que é o padrão quando não especificamos de forma explícita. Neste caso a ação foi imprimir em tela a lista de arquivos que atendem ao critério definido na expressão.

Vamos fazer mais uma busca, desta vez em outro diretório do sistema:

```
daniel@casadocodigo:~$ find /etc/init -name net*
/etc/init/network-interface-container.conf
/etc/init/network-interface-security.conf
/etc/init/network-interface.conf
/etc/init/networking.conf
daniel@casadocodigo:~$
```

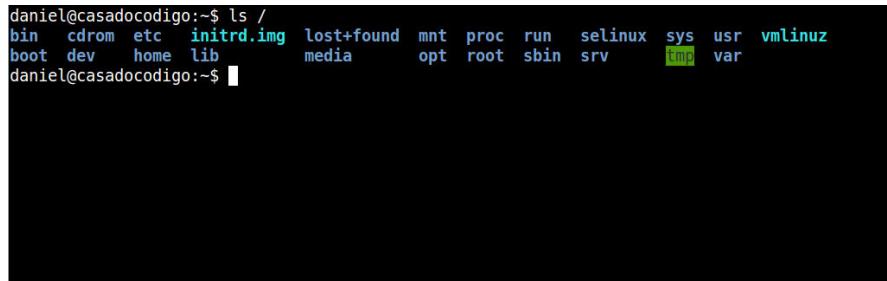
Nesse caso, informamos um caminho diferente para busca no sistema `/etc/init` e especificamos uma nova expressão `-name net*`. O `*` usado em expressões é o mesmo que qualquer coisa; nesse exemplo significa "busque por arquivos que comecem com net e terminem com qualquer coisa".

Teste outra forma de busca com a expressão `-atime`:

```
daniel@casadocodigo:~$ find . -atime +1
./.bash_logout
daniel@casadocodigo:~$
```

No exemplo, o `-atime +1` busca por arquivos que forma modificados há mais de 1 dia. Teste o mesmo comando informando `-1` desta forma ele retornará arquivos modificados a menos de 1 dia:

```
daniel@casadocodigo:~$ find . -atime -1
.
```

```
./.bash_history
./.bashrc
./.lesshst
daniel@casadocodigo:~$
```

Não se preocupe se o resultado no seu computador for diferente, isso é normal pois eu posso ter editado outros arquivos e por isso eles podem aparecer na listagem.

2.7 Um passeio fora do ambiente gráfico

Além do ambiente gráfico no Linux, existe ainda 6 ambientes de linha de comando, conhecidos pelo nome de `TTY` (TeleTYpe) ou ainda terminais virtuais. O `TTY` corresponde ao tipo dos primeiros terminais usados em computadores.

Para ter acesso a esses ambientes, usamos uma combinação das teclas `Ctrl + Alt + F` sendo que *F* corresponde das teclas *F1* a *F6*, a *F7* retorna ao ambiente gráfico.

Faça um teste, pressione a combinação de teclas nesta ordem: `Ctrl + Alt + F1` ou `Ctrl + Alt + F2`:

Figura 2.8: TTY, ambiente nativo de linha de comando

Este é o ambiente padrão fornecido em algumas distribuições Linux para Desktop logo após a instalação, como é o caso do `Slackware` e também é o padrão em servidores.

Para retornar ao modo gráfico faça: `Ctrl + Alt + F7`.

No próximo capítulo, veremos mais sobre edição de arquivos.

Capítulo 3

Manipulando arquivos

Vamos aprender sobre manipulação de arquivos no Shell. Nesse capítulo, conheceremos um pouco sobre o editores de texto no terminal e alguns comandos de leitura de arquivos.

3.1 O editor Vim

O *Vi* é o editor básico disponível em grande parte das distribuições Linux. Atualmente este editor é oferecido com mais recursos e recebe o nome de *Vim* (VI iMproved). O *Vim* é um editor de aspecto simples, porém bastante flexível. Em seguida, veremos o básico sobre ele.

Para começar vamos aprender sobre como entrar e sair do editor. No Ubuntu por padrão ele já vem instalado com o nome de `vi`:

```
daniel@casadocodigo:~$ vi
```

```
~
~
~                         VIM - Vi IMproved
~
~                          version 7.3.429
~                      by Bram Moolenaar et al.
~         Modified by pkg-vim-maintainers@lists.alioth.debian.org
~               Vim is open source and freely distributable
~
~                      Help poor children in Uganda!
~              type  :help iccf<Enter>       for information
~
~              type  :q<Enter>               to exit
~              type  :help<Enter>  or  <F1>  for on-line help
~              type  :help version7<Enter>   for version info
~
~
~
                                                  0,0-1         All
```

Figura 3.1: Tela inicial do vim

Ao entrar no `vi`, ele é aberto direto no modo *visual*. Para editar um arquivo, usamos os modos de *inserção*, *substituição* e *deleção*. Sempre que precisamos voltar ao modo *visual*, usa-se a tecla `Esc`.

Para sair do `vi` use o comando `:q`:

```
~
~
~                         VIM - Vi IMproved
~
~                          version 7.3.429
~                      by Bram Moolenaar et al.
~         Modified by pkg-vim-maintainers@lists.alioth.debian.org
~               Vim is open source and freely distributable
~
~                      Help poor children in Uganda!
~              type  :help iccf<Enter>       for information
~
~              type  :q<Enter>               to exit
~              type  :help<Enter>  or  <F1>  for on-line help
~              type  :help version7<Enter>   for version info
~
~
~
:q
```

Figura 3.2: Saindo do vim

Na sequência veremos como alternar para o modo de *inserção*, escrever em um arquivo, voltar para o *visual*, salvar e sair do editor:

Vamos abrir o `vi` informando o nome de um arquivo para editar:

```
daniel@casadocodigo:~$ vi vim_basico.txt
```

Ao abrir um novo arquivo, o `vi` nos exibe uma tela limpa e simples:

Figura 3.3: Abrindo novo arquivo no vim

Observe na base do editor as informações e o nome do arquivo que abrimos. Vamos alternar para o modo de *inserção* — para isso tecle `i`:

Figura 3.4: Alternando para o modo INSERT

Note que ele informa o modo corrente, no caso INSERT. Nesse momento podemos digitar qualquer coisa no nosso arquivo:

```
Básico de Vim

Aprendendo o básico sobre o editor Vim!_
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                   3,40        All
```

Figura 3.5: Inserindo texto no arquivo

Agora que inserimos um texto, vamos salvar o nosso arquivo. Para isso é ne-

cessário retornar para o modo *visual* e em seguida podemos salvar com o comando `:w`.

Tecle `Esc` para retornar ao modo `visual` e `:w` para salvar:

```
Básico de Vim

Aprendendo o básico sobre o editor Vim!
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:w_
```

Figura 3.6: Salvando o arquivo

Agora que salvamos o arquivo podemos sair do editor com `:q`:

```
Básico de Vim

Aprendendo o básico sobre o editor Vim!
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
:q_
```

Figura 3.7: Saindo do vim após salvar o arquivo

Verifique que depois de salvo, o arquivo foi criado no nosso diretório `home`, não foi preciso criar um arquivo antes:

```
daniel@casadocodigo:~$ ls
vim_basico.txt
daniel@casadocodigo:~$
```

Podemos verificar o conteúdo do arquivo com o `cat`:

```
daniel@casadocodigo:~$ cat vim_basico.txt
Básico de Vim

Aprendendo o básico sobre o editor Vim
daniel@casadocodigo:~$
```

No vim podemos navegar pelo arquivo usando as teclas direcionais ou as teclas `h`, `j`, `k` e `l`:

- `h`, move para a esquerda;
- `j`, move para baixo (próxima linha);
- `k`, move para cima (linha anterior);

- `l`, move para a direita.

Mais comandos que facilitam a movimentação:

- `G` move para a última linha do arquivo;
- `gg` move para a primeira linha do arquivo;
- `0` move o cursor para o início da linha;
- `$` move o cursor para o final da linha.

Alguns comandos básicos para ajudar na inserção de texto:

- `i` para inserir texto antes do cursor;
- `a` para inserir texto depois do cursor;
- `A` para inserir texto no final de uma linha onde se encontra o cursor;
- `o` para adicionar uma linha abaixo da linha atual;
- `O` para adicionar uma linha acima da linha atual.

Temos também comandos para alteração e localização no modo `visual`:

- `x` deleta o caractere onde está o cursor;
- `dd` deleta a linha atual;
- `u` desfaz a última modificação no arquivo;
- `yy` copia uma linha, `p` cola o que foi copiado;
- `/palavra` busca pela palavra ou caractere em todo o arquivo.

A variedade de comandos e combinações para esse editor é muito grande, teríamos um livro apenas sobre ele. Por isso é interessante ter o `vimbook`, um guia em português, ótimo para consultas:

https://code.google.com/p/vimbook/

Se preferir um screencast, o Fabio Akita disponibilizou gratuitamente um vídeo com 1 hora de duração sobre o vim para iniciantes:

http://blip.tv/akitaonrails/screencast-come-ando-com-vim-6075050

3.2 O editor Nano

O `nano` é mais uma alternativa para edição de textos no terminal, e está disponível em quase todas as distribuições linux, assim como o `vim`.

O nano acaba sendo uma boa escolha para iniciantes por ser mais fácil de usar e possuir uma interface bastante intuitiva.

Vamos abrir o `nano` e conhecer um pouco sobre esse editor:

Figura 3.8: Tela inicial do nano

Observando o menu inferior do editor `nano` notamos várias funções. Vale lembrar que ^ é o mesmo que **Ctrl**, sendo assim `^G` significa `Ctrl + G`.

Alguns comandos básicos do `nano`:

- `^G` apresenta a tela de ajuda sobre mais comandos do editor;
- `^X` sai do editor; se o arquivo não estiver salvo será solicitado para salvar antes de sair;
- `^O` salva um arquivo;
- `^W` busca por uma palavra dentro do arquivo.

Vamos abrir o arquivo que criamos com o `vim`, editar e salvar para entender na prática como funciona o `nano`.

Com o editor aberto. tecle ^R. Será solicitado o nome do arquivo que desejamos editar, no nosso caso vim_basico.txt:

Figura 3.9: Abrindo o arquivo vim_basico.txt com o nano

Agora que temos o arquivo aberto, vamos editar:

Figura 3.10: Editando o arquivo com o nano

Vamos salvar nosso arquivo editado com o atalho ^O. Será solicitado o nome do arquivo caso queira criar um novo arquivo:

Figura 3.11: Salvando o arquivo com o nano

Após salvar o arquivo, tecle `^X` para sair do editor e verifique que agora temos um novo arquivo com o nome `nano_basico.txt`:

```
daniel@casadocodigo:~$ nano
daniel@casadocodigo:~$ ls
nano_basico.txt  vim_basico.txt
daniel@casadocodigo:~$
```

Como podemos ver, editar textos com o `nano` é bem simples. A seguir veremos alguns comandos que podem ajudar na edição de arquivos de forma rápida.

3.3 Trabalhando com o Cat

O comando `cat`, que já vimos um pouco anteriormente, é normalmente utilizado para exibir o conteúdo de arquivos, mas ele possui recursos para ir além disso.

Para conhecermos melhor o `cat`, vamos criar um arquivo chamado `agenda` e inserir alguns dados como nome e perfil no Twitter. Criaremos o arquivo usando o próprio `cat`, com o parâmetro `>`. Desta forma, o `cat` vai ler os dados digitados no teclado e inserir no arquivo indicado.

Faça `cat > agenda` e tecle *Enter*. Em seguida digite as informações, e no final tecle `Ctrl + d`. Isto vai encerrar a digitação e salvar os dados no arquivo `agenda`:

```
daniel@casadocodigo:~$ cat > agenda
Daniel Romero   @infoslack
```

```
Paulo Silveira  @paulo_caelum
Nando Vieira    @fnando
Adriano Almeida @adrianoianoalmeida7
Vinicius Baggio @vinibaggio
^D
```

Para conferir, exiba o conteúdo do arquivo e verifique que os dados foram salvos:

```
daniel@casadocodigo:~$ cat agenda
Daniel Romero   @infoslack
Paulo Silveira  @paulo_caelum
Nando Vieira    @fnando
Adriano Almeida @adrianoalmeida7
Vinicius Baggio @vinibaggio
daniel@casadocodigo:~$
```

O `cat` disponibiliza opções para exibir <*TAB*> no arquivo `-t`, o fim das linhas de um arquivo, ou seja, onde demos *ENTER* `-e`. Faça um teste com as duas opções:

```
daniel@casadocodigo:~$ cat -et agenda
Daniel Romero^I@infoslack$
Paulo Silveira^I@paulo_caelum$
Nando Vieira^I@fnando$
Adriano Almeida^I@adrianoalmeida7$
Vinicius Baggio^I@vinibaggio$
daniel@casadocodigo:~$
```

Note que foram exibidos caracteres `^`, que representam o <*TAB*>, e o `$`, que representa o fim da linha ou <*ENTER*>. Existe também a opção de enumerar as linhas `-n`:

```
daniel@casadocodigo:~$ cat -n agenda
     1  Daniel Romeroero @infoslack
     2  Paulo Silveira   @paulo_caelum
     3  Nando Vieira     @fnando
     4  Adriano Almeida  @adrianoalmeida7
     5  Vinicius Baggio  @vinibaggio
daniel@casadocodigo:~$
```

Um uso muito interessante do `cat` é para concatenar arquivos. Ele pode pegar o conteúdo de vários arquivos e redirecionar para um outro. Lembra do arquivo que criamos com o `vim`? Vamos usá-lo para testar esse recurso do `cat`, pegaremos os

dados da agenda e do arquivo criado com o `vim`, e vamos concatenar em um novo arquivo.

Antes, verifique o conteúdo de cada um:

```
daniel@casadocodigo:~$ cat vim_basico.txt
Básico de Vim

Aprendendo o básico sobre o editor Vim
daniel@casadocodigo:~$
daniel@casadocodigo:~$ cat agenda
Daniel Romero   @infoslack
Paulo Silveira  @paulo_caelum
Nando Vieira    @fnando
Adriano Almeida @adrianoalmeida7
Vinicius Baggio @vinibaggio
daniel@casadocodigo:~$
```

Vamos à instrução de concatenação de arquivos. Para isso faça o seguinte: `cat vim_basico.txt agenda > concatenando.txt`:

```
daniel@casadocodigo:~$ cat vim_basico.txt agenda > concatenando.txt
daniel@casadocodigo:~$
```

Em seguida, verifique o conteúdo do novo arquivo gerado pelo `cat`:

```
daniel@casadocodigo:~$ cat concatenando.txt
Básico de Vim

Aprendendo o básico sobre o editor Vim
Daniel Romero   @infoslackslack
Paulo Silveira  @paulo_caelum
Nando Vieira    @fnando
Adriano Almeida @adrianoalmeida7
Vinicius Baggio @vinibaggio
daniel@casadocodigo:~$
```

3.4 Exibindo o início e o fim de arquivos

É interessante saber sobre dois comandos com a função de exibir o conteúdo de um arquivo do `cat`: são eles o `head` e `tail`.

O `head` mostra o início dos dados em um arquivo. Por padrão ele exibe as 10 primeiras linhas do arquivo, mas utilizando a opção `-n` podemos limitar o número de linhas:

```
daniel@casadocodigo:~$ head -n 3 concatenando.txt
Básico de Vim

Aprendendo o básico sobre o editor Vim
daniel@casadocodigo:~$
```

Já o `tail` mostra o final dos dados do arquivo e possui a mesma opção para limitar a quantidade de linhas, `-n`:

```
daniel@casadocodigo:~$ tail -n 5 concatenando.txt
Daniel Romero   @infoslackslackack
Paulo Silveira  @paulo_caelum
Nando Vieira    @fnando
Adriano Almeida @fnandoadrianoalmeida7
Vinicius Baggio @vinibaggio
daniel@casadocodigo:~$
```

Lembre-se de sempre consultar a documentação dos comandos com o `man`!

No capítulo a seguir veremos como fazer a compreesão e descompressão de arquivos.

Capítulo 4

Compactação e descompactação de arquivos

O Linux por padrão possui vários utilitários de compressão de arquivos. Compactar arquivos e diretórios é uma boa prática para realização de *backups*. Neste capítulo veremos alguns comandos de compressão.

4.1 Conhecendo o tar

O `tar` é excelente para agrupar vários arquivos em apenas um. Isso, na prática, evita que precisemos fazer várias transferências de arquivos entre computadores, enviando apenas um único arquivo. Isso vai aparecer **muito** no decorrer do seu uso do Linux. É importante conhecê-lo bem.

Basicamente existem três tipos de arquivos que são utilizados com o `.tar`: `tar.gz` ou `.tgz` e `.tar.bz2`.

Ao utilizarmos o `.tar`, ele realiza uma cópia sem compactação dos arquivos

passados como parâmetro, ou seja ele junta todos os arquivos em um único arquivo sem compactar.

Algumas opções do `tar`:

- `-c` cria um novo arquivo .tar e adiciona os arquivos a serem compactados;
- `-f` indica que o destino é um arquivo em disco;
- `-v` exibe o nome de cada arquivo compactado;
- `-x` extrai os arquivos agrupados no arquivo .tar.

Para mais opções, lembre-se de usar o `man`!

Vamos testar o `.tar` na prática, juntaremos todos os nossos arquivos que terminam em `.txt` em apenas um. Para isso usaremos as opções `-c` e `-f`:

```
daniel@casadocodigo:~$ ls
agenda  concatenando.txt  nano_basico.txt  vim_basico.txt
daniel@casadocodigo:~$ tar -cf backup.tar *.txt
daniel@casadocodigo:~$ ls
agenda  backup.tar  concatenando.txt  nano_basico.txt  vim_basico.txt
daniel@casadocodigo:~$
```

Note que foi criado um arquivo chamado `backup.tar`, que representa a união de todos os arquivos com extensão `.txt`.

Vamos fazer o reverso desta união, separando os arquivos usando as opções `-x`, `-v` e `-f`. Criaremos um novo diretório para melhor visualizar:

```
daniel@casadocodigo:~$ mkdir backup
daniel@casadocodigo:~$ mv backup.tar backup/
daniel@casadocodigo:~$ cd backup/
daniel@casadocodigo:~/backup$ ls
backup.tar
daniel@casadocodigo:~/backup$ tar xvf backup.tar
concatenando.txt
nano_basico.txt
vim_basico.txt
daniel@casadocodigo:~/backup$ ls
backup.tar  concatenando.txt  nano_basico.txt  vim_basico.txt
daniel@casadocodigo:~/backup$
```

O que acabamos de fazer foi criar uma pasta chamada `backup`, mover o arquivo `backup.tar` para dentro da pasta criada e fazer o processo reverso para separar os arquivos.

O tipo .tar.gz

O `.tar.gz` consiste em dois processos diferentes e interligados, sendo que primeiro ele cria um arquivo `.tar` e, em seguida, compacta utilizando o formato `.gz`. Neste caso, além de gerar um único arquivo com o `.tar`, ele será compactado, reduzindo assim o seu tamanho.

Ainda dentro da pasta `backup`, vamos testar a criação do nosso arquivo dessa vez compactado com `.tar.gz`:

```
daniel@casadocodigo:~/backup$ ls
backup.tar  concatenando.txt  nano_basico.txt  vim_basico.txt
daniel@casadocodigo:~/backup$ tar zcvf backup.tar.gz *.txt
concatenando.txt
nano_basico.txt
vim_basico.txt
daniel@casadocodigo:~/backup$ ls
backup.tar     concatenando.txt  vim_basico.txt
backup.tar.gz  nano_basico.txt
daniel@casadocodigo:~/backup$
```

Note que agora temos o `backup.tar.gz`, o mesmo arquivo unido `.tar`, mas compactado com `.gz`. Podemos verificar o tamanho dos arquivos com o comando `du` para melhor entender o real efeito da compactação:

```
daniel@casadocodigo:~/backup$ du -h backup.tar backup.tar.gz
12K   backup.tar
4.0K  backup.tar.gz
daniel@casadocodigo:~/backup$
```

Veja que usando o `.tar.gz` o arquivo foi reduzido para 4 kbytes! Para descompactar arquivos `.tar.gz` usa-se as opções `zxvf`:

```
daniel@casadocodigo:~/backup$ tar zxvf backup.tar.gz
concatenando.txt
nano_basico.txt
vim_basico.txt
daniel@casadocodigo:~/backup$
```

O tipo .tar.bz2

Com o `.tar.bz2` ocorre o mesmo caso que com o `.tar.gz`: a compactação com o formato `.bz2` é mais eficiente que o `.gz`, porém é mais lenta na hora de gerar a cópia.

Vamos criar um arquivo compactado com `.tar.bz2` utilizando as opções `jcvf` e conferir a diferença entre os tamanhos com o comando `du`:

```
daniel@casadocodigo:~/backup$ tar jcvf backup.tar.bz2 *.txt
concatenando.txt
nano_basico.txt
vim_basico.txt
daniel@casadocodigo:~/backup$ ls
backup.tar      backup.tar.gz     nano_basico.txt
backup.tar.bz2  concatenando.txt  vim_basico.txt
daniel@casadocodigo:~/backup$ du -h backup.tar*
12K   backup.tar
4.0K  backup.tar.bz2
4.0K  backup.tar.gz
daniel@casadocodigo:~/bashackup$
```

Não houve diferença entre os tamanhos nos formatos `.gz` e `.bz2`, já que estamos compactando pequenos arquivos texto. Porém é possível notar a diferença em arquivos maiores, por exemplo uma imagem *ISO* de um *DVD*.

Para fazer a descompactação de formatos `.bz2` usamos as opções `jxvf`:

```
daniel@casadocodigo:~/backup$ tar jxvf backup.tar.bz2
concatenando.txt
nano_basico.txt
vim_basico.txt
daniel@casadocodigo:~/backup$
```

Podemos notar que, sempre que vamos compactar, a opção `c` é utilizada; e ao realizar a descompactação usamos a opção `x`. Já para o formato `.tar.gz`, usamos a opção `z`, e a opção `j` para o formato `.tar.bz2`.

Pode parecer um pouco complicado memorizar todas essas opções no inicio, mas com a prática isso acaba se tornando natural.

4.2 Conhecendo o gzip/gunzip

Este é o programa de compressão mais utilizado no Linux. O `gzip` gera um arquivo no formato `.gz`. Sua forma de uso para compactar é bastante simples:

```
daniel@casadocodigo:~/backup$ gzip -c -r *.txt > backup.gz
daniel@casadocodigo:~/backup$ ls
backup.gz  concatenando.txt  nano_basico.txt  vim_basico.txt
daniel@casadocodigo:~/backup$
```

As opções `-c` e `-r` respectivamente informam que queremos criar um arquivo e compactar. Ao executarmos a compactação de vários arquivos utilizando o `gzip`, os arquivos serão concatenados em um só e em seguida comprimidos no formato `.gz`.

Para descompactar usamos o `gunzip`, que pode ser utilizado com as opções `-c` e `-v` para exibir o conteúdo a ser descompactado e informações sobre o nível de compactação dos arquivos:

```
daniel@casadocodigo:~/backup$ gunzip backup.gz
daniel@casadocodigo:~/backup$
```

No exemplo, ao executar a descompactação com `gunzip`, o nosso arquivo `backup.gz` deixará de existir pois por padrão o `gunzip` extrai o arquivo comprimido no formato `.gz` e exclui o arquivo compacto.

4.3 Conhecendo o zip/unzip

O comando `zip`, como o nome já sugere, cria um arquivo compacto no formato `.zip`, enquanto o `unzip` faz o trabalho inverso, descompactando.

A forma de uso é muito simples:

```
daniel@casadocodigo:~/backup$ zip backup.zip *.txt
  adding: concatenando.txt (deflated 27%)
  adding: nano_basico.txt (deflated 44%)
  adding: vim_basico.txt (deflated 16%)
daniel@casadocodigo:~/backup$ ls
backup.zip  concatenando.txt  nano_basico.txt  vim_basico.txt
daniel@casadocodigo:~/backup$
```

Pacotes padrões em um Ubuntu

Tente este comando agora. Há uma grande chance de você não conseguir executá-lo.

Por quê? Porque ele não está instalado! O Ubuntu e as outras distros em geral trazem já instalados um grupo grande de aplicativos, mas obviamente não todos. O Zip é um deles. Em breve, veremos como procurar e instalar os pacotes do Linux.

Para descompactar, use o comando `unzip`:

```
daniel@casadocodigo:~/backup$ unzip backup.zip
Archive:  backup.zip
  inflating: concatenando.txt
  inflating: nano_basico.txt
  inflating: vim_basico.txt
daniel@casadocodigo:~/backup$
```

É importante não confundir o comando `gzip` com o `zip` e o `gunzip` com o `unzip`!

Capítulo 5

Entendendo a estrutura de diretórios

Vamos conhecer mais sobre a estrutura de diretórios Linux e entender o seu funcionamento. A estrutura de diretórios armazena arquivos de forma hierárquica, de maneira que o usuário não precisa conhecer os detalhes técnicos do sistema de armazenamento. Mas para poder navegar e obter informações é preciso entender esta estrutura.

5.1 A estrutura de diretórios

Cada diretório do sistema possui arquivos que são mantidos seguindo regras definidas pela **FHS** (Filesystem Hierarchy Standard). No Ubuntu a estrutura de diretórios segue o padrão LSB (Linux Standard Base), que por sua vez segue a especificação do FHS. Esses padrões são importantes pois ajudam a manter a compatibilidade entre as variações de distribuições Linux.

Como vimos nos primeiros capítulos, um diretório é um local onde guardamos arquivos no sistema, também conhecidos por pastas. Vamos relembrar a nossa primeira listagem de diretórios, que pode ser chamada de *árvore*:

```
daniel@casadocodigo:~$ ls /
bin   home        media  Repos  selinux  usr
boot  initrd.img  mnt    root   srv      vagrant
dev   lib         opt    run    sys      var
etc   lost+found  proc   sbin   tmp      vmlinuz
daniel@casadocodigo:~$
```

Nesta estrutura, o FHS determina que obrigatoriamente uma distribuição Linux deve conter 14 diretórios. Veremos cada um deles.

Quando listamos `ls /` estamos verificando o conteúdo do diretório `raiz`. `/` é um diretório — podemos dizer que é o principal diretório do sistema, pois nele ficam todos os outros. Quem vem do Windows pode pensar no `/` como o `c:` ou no ícone `Meu Computador`. Todo diretório dentro do raiz (`/`) é chamado de subdiretório.

O diretório `/bin` armazena arquivos executáveis binários, que são os comandos base para a execução do sistema, por exemplo o `ls` e o `cd`. Esse diretório é público, ou seja, qualquer usuário pode usar os executáveis que estão lá.

O `/boot` contém arquivos de inicialização do sistema, dentre os quais está o gerenciador de `boot` do sistema. Ele é um aplicativo que carrega o sistema operacional durante a inicialização.

O diretório `/dev` mantém o caminho dos dispositivos instalados no sistema. Todo o hardware reconhecido pelo sistema é representado por um arquivo nesse diretório, por exemplo, disco rígido e placa de vídeo.

Em `/etc` ficam os arquivos de configuração do sistema, scripts de inicialização, configurações padrão para usuários e arquivos de configuração de programas que são instalados. Veremos muito esse diretório quando estivemos instalando aplicativos.

O `/lib` contém as bibliotecas e módulos do kernel que são essenciais para o funcionamento do sistema. As bibliotecas são funções compartilhadas que podem ser usadas por vários programas.

`/media` é o diretório responsável por manter os *pontos de montagem*, ou seja, quando inserimos um pen drive é neste diretório que ele ficará disponível temporariamente enquanto usamos.

O `/mnt` é utilizado para *montagem temporária* de sistemas de arquivos, isto é, um hd ou pen drive. Este diretório pode ser usado da mesma forma que o `/media`.

No diretório `/opt` é onde normalmente instalamos programas que não fazem parte oficialmente da distribuição. Por exemplo, o `google chrome`.

Em `/sbin` ficam os comandos utilizados para inicialização, reparação e restauração do sistema. É um diretório de comandos essenciais, mas com a diferença de que apenas um usuário pode usar, o `root`. Veremos mais sobre esse usuário no próximo capítulo.

O `/srv` mantém dados de serviços disponíveis pelo sistema e pode ser acessado de forma geral (por todos os usuários), por exemplo, `web server`.

No `/tmp` ficam armazenados arquivos temporários, informações que devem ser mantidas até o fim de uma operação, como um download em andamento ou arquivos de cache de vídeos do Youtube.

Em `/usr` são mantidos programas que não são essenciais para o funcionamento do sistema. Programas instalados pelo usuário, como editores, programas gráficos, gerenciadores de janelass são exemplos disso.

O diretório `/var` contém arquivos de dados variáveis, ou seja, arquivos que podem aumentar de tamanho, como arquivos de log, arquivos de bancos de dados e mensagens de e-mail.

5.2 Os diretórios opcionais

Os diretórios `/home` e `/root` são opcionais — eles podem existir no sistema mas não obrigatoriamente com estes nomes, apesar de serem assim com frequência!

O diretório `/home` armazena os diretórios e arquivos dos usuários cadastrados no sistema, por exemplo `/home/daniel`. Ele poderia ser chamado por outro nome como `/minha_pasta`, por exemplo, e isso não afetaria em nada a estrutura do sistema.

Já o diretório `/root` é a pasta pessoal do superusuário `root`, sobre o qual veremos mais detalhes no capítulo seguinte.

5.3 Os diretórios /proc e /sys

O diretório `/proc` contém arquivos temporários de processos em execução no sistema. Em outras palavras, é um diretório virtual usado pelo kernel. Nele são mantidos configurações atuais do sistema e dados estatísticos. Mais adiante veremos detalhes sobre processos.

Assim como o diretório `/proc`, o `/sys` armazena quase o mesmo conteúdo porém de forma mais organizada para podermos administrar.

Á medida que formos avançando, vamos conferir na prática alguns dos principais diretórios vistos aqui.

Capítulo 6

Administração de usuários

O Linux é um sistema multiusuário, ou seja, que pode ser usado por vários usuários simultaneamente, sem que um interfira nas atividades do outro, nem consiga alterar seus arquivos. Um bom exemplo para entender melhor seria um servidor web compartilhado, baseado em Linux, que hospeda alguns milhares de sites, cada um administrado por um usuário diferente.

As restrições implementadas no sistema de permissões são muito eficientes. Esse esquema de permissões é fundamental para o funcionamento do sistema por completo. As permissões consistem em um conjunto de três regras: *leitura*, *escrita* e *execução*. É graças a esse esquema de permissões que não vemos sistemas Linux infectados por malwares ou vírus.

Neste capítulo, aprenderemos sobre gestão de usuários e vamos entender detalhadamente como funcionam as permissões.

6.1 Gerenciando usuários

Para entender sobre o controle de usuários, vamos dividir esta seção em três categorias:

1) Super Usuário (sudo) ou Administrador;

2) Usuário de Sistema;

3) Usuário Comum.

Usuário Administrador e sudo

No sistema, esse usuário é chamado de `root`. Ele é responsável por controlar todo o sistema e não possui restrições. Sempre que executamos algum programa ou tarefa que necessite de poderes administrativos, precisamos do `root`, que é chamado por meio do comando `sudo`.

Por exemplo, sempre que formos instalar um programa ou atualizar todo o sistema, usaremos o comando `sudo` para ter as permissões de `root` e conseguir efetuar essas tarefas.

Veja o que acontece ao tentar executar uma tarefa que precisa das permissões do usuário `root`:

```
daniel@casadocodigo:~$ apt-get update
E: Could not open lock file
  /var/lib/apt/lists/lock - open (13: Permission denied)
E: Unable to lock directory /var/lib/apt/lists/
E: Could not open lock file
  /var/lib/dpkg/lock - open (13: Permission denied)
E: Unable to lock the administration directory
  (/var/lib/dpkg/), are you root?
daniel@casadocodigo:~$
```

Podemos notar várias mensagens de alerta informando que a permissão para a tarefa foi negada e, no final, ele pergunta se somos o usuário `root`. O correto seria executar com o comando `sudo` para que o nosso usuário seja tratado com os privilégios do usuário `root`.

Veja o que acontece ao executarmos a mesma instrução usando o comando `sudo`:

```
daniel@casadocodigo:~$ sudo apt-get update
[sudo] password for daniel:
```

Note que foi solicitada uma senha para o nosso usuário, que é a mesma que cadastramos no processo de instalação. Ao confirmá-la, o comando `apt-get update` funcionará normalmente, pois o nosso usuário faz parte do mesmo `grupo` do usuário `root`. Entenderemos mais adiante sobre *grupos de usuários*.

Não se preocupe ainda com o exemplo do comando `apt-get`, veremos o que ele faz no próximo capítulo.

Usuários de sistema

São usuários que não necessitam logar no sistema – eles existem para controlar serviços e normalmente não possuem senhas. Um bom exemplo é o usuário `www-data`, que pode ser usado para administrar servidores web como Apache e Nginx.

Veremos mais sobre usuários de sistema nos capítulos sobre administração e serviços.

Usuários comuns

São as contas criadas para o utilizadores do sistema. Essencialmente, eles podem executar tarefas básicas como criar e editar documentos, navegar na internet, assistir vídeos etc. A conta `daniel` é um exemplo de usuário comum, diferente da conta `root` que é utilizada para administrar o sistema.

6.2 Permissões

Nesta seção abordaremos o suficiente para termos uma referência sobre permissões de arquivos e usuários. As permissões são opções que permitem um usuário ter controle de acesso, leitura, gravação e execução de arquivos.

Existem três tipos de permissões: `r` *(leitura)*, `w` *(escrita)* e `x` *(execução)*.

Arquivos disponíveis somente para leitura podem ser abertos e ter seu conteúdo visualizado, assim como pastas com o mesmo tipo de permissão podem ter os arquivos listados.

Arquivos que possuem permissão de escrita podem ser alterados – o usuário, neste caso, tem permissão para editar um arquivo e até apagar. Em diretórios é possível criar um novo arquivo.

A permissão de execução dá a possibilidade de executar um arquivo como um programa, por exemplo um `script` de instalação de um aplicativo.

Para listar as permissões de um arquivo ou diretório, podemos usar o comando `ls`, seguido da opção `-l` (long listing). Assim teremos uma listagem detalhada dos arquivos:

```
daniel@casadocodigo:~$ ls -l
total 20
-rw-rw-r-- 1 daniel daniel  136 Oct  2 00:49 agenda
drwxrwxr-x 2 daniel daniel 4096 Oct  2 18:49 backup
-rw-rw-r-- 1 daniel daniel  192 Oct  2 01:12 concatenando.txt
-rw-rw-r-- 1 daniel daniel   98 Sep 28 16:35 nano_basico.txt
-rw-rw-r-- 1 daniel daniel   56 Sep 28 01:50 vim_basico.txt
daniel@casadocodigo:~$
```

No começo, isso pode parecer grego para nós. É comum uma pequena dificuldade até se acostumar com tantas letrinhas e abreviações. Vamos entender detalhadamente o significado da listagem exibida:

- `-`: significa "desabilitado" ou permissão negada;
- `r`: permissão de leitura, apenas lê um arquivo;
- `w`: permissão de escrita, pode escrever em arquivos e diretórios;
- `x`: permissão para executar um arquivo.

A aplicação de permissões aos arquivos está ligada a três entidades, que são classes de acesso aos arquivos:

- `u`: usuário dono do arquivo;
- `g`: grupo a que o arquivo pertence;
- `o`: outros usuários que não sejam o dono nem pertençam ao grupo.

Quando listamos com `ls -l`, os *10 primeiros caracteres* indicam o modo do arquivo:

```
-rw-rw-r-- 1 daniel daniel  136 Oct  2 00:49 agenda
```

Figura 6.1: Modo do arquivo está sublinhado em vermelho

O primeiro caractere indica se é um diretório, link ou um arquivo. Os outros *9 caracteres formam três grupos de três caracteres*.

O primeiro caractere de cada grupo representa a permissão de *leitura*, `r`. O segundo caractere representa a permissão de *escrita*, `w`. O terceiro caractere representa a permissão de *execução*, `x`. Quando o caractere `-` *(hífen)* é exibido, significa que a permissão está desligada.

Para entender melhor os grupos veja a imagem seguinte da mesma listagem:

Figura 6.2: Os três grupos

Analisando com calma é possível perceber que as permissões para o `dono` do arquivo são de leitura e escrita. As permissões para o `grupo` ao qual o arquivo pertence são também de leitura e escrita, e as permissões de `outros` são de apenas leitura. Observamos que `daniel` é o nome do dono do arquivo e também é o nome do grupo a qual o arquivo pertence.

Para evitar confusão, mudaremos o nome do grupo mais adiante.

No ambiente gráfico é possível verificar as permissões de um arquivo: para isso, clique com o botão direito do mouse no arquivo, depois em `propriedades` e, em seguida, na aba `permissões`:

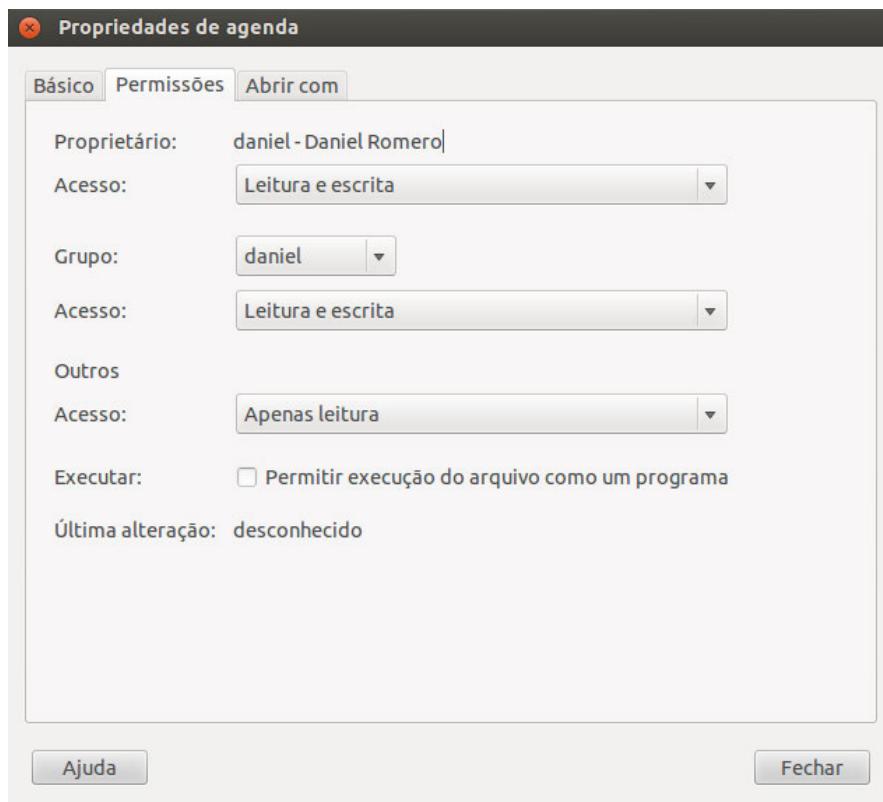



Figura 6.3: Permissões de um arquivo no ambiente gráfico

6.3 Atribuindo permissões

O `chmod` é o comando para atribuir permissões de arquivos. Basicamente, ele atua diretamente nos níveis *dono*, *grupo* e *outros*.

A sintaxe de uso do `chmod` é a seguinte:

```
chmod [opções] modo arquivo
```

O `modo` pode ser representado utilizando notação simbólica ou octal, veremos as duas formas. A seguir, uma lista das opções para o comando:

- `-c`: lista informações sobre os arquivos modificados;
- `-f`: ignora mensagens de erro;

- `-R`: modo recursivo, altera as permissões de todos os arquivos e diretórios;
- `-v`: lista de forma detalhada as alterações de atributo.

Antes de prosseguir é importante entendermos sobre **bits de atributo**!

Bits de atributo

Bits de atributo são um controle adicional às permissões de arquivos. As entidades *dono*, *grupo* e *outros* têm suas configurações setadas pelo dono do arquivo. Essas configurações podem ser representadas também pelos *bits de atributo*, ou seja, assim como existem três entidades, também existem três *bits de atributo*, que podem estar ligados ou desligados:

Bit SetUID permite que um arquivo seja executado da mesma forma que é executado pelo dono.

Bit SetGID permite que um arquivo seja executado como se fosse parte do grupo de seu dono. Quando um arquivo é criado em um diretório que possui este bit ativo, ele é criado com o mesmo grupo do diretório.

Bit Sticky ativa uma proteção para o arquivo e não permite que ele seja apagado, a não ser pelo próprio dono.

A representação simbólica destes *bits* é lida da esquerda para a direita através das siglas `SST`, onde o primeiro *S* representa o *SetUID*, o segundo *S* representa o *SetGID* e o *T* representa o *Sticky*. Para indicar se um *bit* está ativo ou inativo, usa-se *(1 ou +)* para ativo e *(0 ou -)* para inativo.

Os bits de proteção definem proteções básicas de um arquivo que, como vimos anteriormente, são: *leitura*, *escrita* e *execução*. Ativar ou não estes bits é o que define as permissões de quem poderá trabalhar com o arquivo, ou seja: *editar*, *deletar* ou *executar*.

Bits de proteção são divididos em três grupos de três bits, sendo que cada grupo possui três bits e cada bit representa uma permissão. Ficou confuso? Vamos ver uma representação para entender melhor:

000 000 000 as permissões de cada entidade são representadas por cada grupo de três zeros e são lidas sempre da esquerda para a direita.

Quando listamos arquivos com o comando `ls -l`, conseguimos visualizar as mesmas permissões de outra maneira: **rwx rwx rwx**.

Os tipos de notações

Agora que já vimos o que são os *bits de atributo* e de *proteção*, veremos como converter entre notações binária, simbólica e octal.

Juntando os bits de atributo e de proteção temos um total de 12 bits, sendo 3 de atributos e 9 de proteção. Veja a representação binária e simbólica: **000 000 000 000** ou **sst rwx rwx rwx**.

A notação octal converte cada grupo de três bits em um caractere que vai de 0 a 7.

```
Grupos:       Dono  |   Grupo   | Outros
Simbólica:    rwx       r-x       r--
Binária:      111       101       100
Octal:         7         5         4
```

Voltando para o `chmod`, vamos à prática. Primeiro, com o comando `ls -l`, vamos listar as informações sobre permissão do arquivo `agenda`:

```
daniel@casadocodigo:~$ ls -l agenda
-rw-rw-r-- 1 daniel daniel 136 Oct  2 00:49 agenda
daniel@casadocodigo:~$
```

Podemos notar que o arquivo possui permissão de leitura e escrita para o dono e grupo, e permissão de apenas leitura para outros. Fazendo uso do `chmod` vamos alterar as permissões deste arquivo:

```
daniel@casadocodigo:~$ chmod u=rw,g=rw,o=rw agenda
daniel@casadocodigo:~$ ls -l agenda
-rw-rw-rw- 1 daniel daniel 136 Oct  2 00:49 agenda
daniel@casadocodigo:~$
```

O que fizemos foi manter as permissões para dono e grupo e setar uma nova permissão para outros. Agora o nosso arquivo possui a permissão de leitura e escrita para *outros*.

Lembre-se que o *u* representa o usuário dono, o *g* representa o grupo e o *o* representa outros. A mesma permissão poderia ter sido executada na forma octal:

```
daniel@casadocodigo:~$ chmod 666 agenda
daniel@casadocodigo:~$ ls -l agenda
-rw-rw-rw- 1 daniel daniel 136 Oct  2 00:49 agenda
daniel@casadocodigo:~$
```

Teste o `chmod` usando a opção `-v` e veja a informação que ele retorna:

```
daniel@casadocodigo:~$ chmod -v 664 agenda
mode of `agenda` changed from 0666 (rw-rw-rw-) to 0664 (rw-rw-r--)
daniel@casadocodigo:~$
```

Retornamos a permissão anterior usando o formato octal. A opção `-v` do `chmod` nos mostra duas notaçõesm, de modo que fica fácil comparar e entender o que estamos fazendo.

Analise a tabela de conversão entre as notações para os bits de proteção. Usaremos a notação octal daqui pra frente por ser mais concisa:

Binário	Simbólico	Octal
000	---	0
001	--x	1
010	-w-	2
011	-wx	3
100	r--	4
101	r-x	5
110	rw-	6
111	rwx	7

Figura 6.4: Notações para os bits de proteção

6.4 Criando grupos

Agora que já entendemos um pouco sobre usuários, grupos e permissões, vamos praticar e conhecer mais alguns comandos.

Sabemos que um usuário deve pertencer a um grupo e pode ser adicionado a outros grupos. Para criar um novo grupo, usamos o comando `addgroup` – criaremos um grupo chamado `suporte`:

```
daniel@casadocodigo:~$ sudo addgroup suporte
Adding group `suporte` (GID 1005) ...
Done.
daniel@casadocodigo:~$
```

Note que, ao criar o grupo automaticamente, ele recebe o seu número GID.

6.5 Criando usuários

Para criar usuários, existe o comando adduser, que pode também adicionar usuários em grupos e até criar novos grupos.

adduser paulo irá adicionar o usuário paulo solicitando as informações adicionais como: nome, senha, grupo, pasta home...:

```
daniel@casadocodigo:~$ sudo adduser paulo
Adding user `paulo` ...
Adding new group `paulo` (1007) ...
Adding new user `paulo` (1003) with group `paulo` ...
Creating home directory `/home/paulo` ...
Copying files from `/etc/skel` ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for paulo
Enter the new value, or press ENTER for the default
Full Name []:
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n] Y
```

Note todas as opções solicitadas ao criar um novo usuário. Após preencher ou confirmar somente as que achar necessário, o usuário paulo será criado no sistema:

```
daniel@casadocodigo:~$ id paulo
uid=1003(paulo) gid=1007(paulo) groups=1007(paulo)
daniel@casadocodigo:~$ ls /home/
daniel  paulo
daniel@casadocodigo:~$
```

Utilize o comando id sempre que quiser obter informações sobre um usuário do sistema. Ele irá retornar UID, GID e os grupos aos quais o usuário pertence.

Agora que já criamos um novo grupo e um novo usuário, vamos adicionar o usuário paulo ao grupo suporte. Para isso, utilize o comando addgroup que,

além de criar um novo grupo como vimos anteriormente, também pode adicionar usuários a outros grupos:

```
daniel@casadocodigo:~$ sudo addgroup paulo suporte
Adding user `paulo` to group `suporte` ...
Adding user paulo to group suporte
Done.
daniel@casadocodigo:~$
```

O retorno do comando já nos informa que o usuário `paulo` foi adicionado ao grupo `suporte`. Podemos verificar com o comando `groups` ou `id`:

```
daniel@casadocodigo:~$ groups paulo
paulo : paulo suporte
daniel@casadocodigo:~$ id paulo
uid=1003(paulo) gid=1007(paulo) groups=1007(paulo),1005(suporte)
daniel@casadocodigo:~$
```

Um novo usuário pode ser criado e de imediato já possuir um grupo de nossa escolha. O `adduser` poderia fazer isso recebendo como argumentos o nome do usuário e o nome do grupo: `adduser [usuário] [grupo]`. E para remover usuário e grupo poderíamos usar o comando `deluser`, que tem a seguinte sintaxe: `deluser [usuário] [grupo]`.

6.6 Alterando grupos

Como vimos no inicio do capítulo, cada arquivo e diretório possui um dono e um grupo. Tanto o dono como o grupo ao qual um arquivo ou diretório pertence podem ser alterados. Para isso, usamos os comandos `chown` e `chgrp`.

Para testá-los, vamos usar o arquivo `agenda` que criamos anteriormente:

```
daniel@casadocodigo:~$ ls -l agenda
-rw-rw-r-- 1 daniel daniel 136 Oct  2 00:49 agenda
daniel@casadocodigo:~$
```

O dono do arquivo é o usuário `daniel`, assim como o grupo ao qual o arquivo pertence também é `daniel`. Vamos alterar o grupo a que o arquivo pertence para `suporte`:

```
daniel@casadocodigo:~$ sudo chgrp suporte agenda
daniel@casadocodigo:~$ ls -l agenda
```

```
-rw-rw-r-- 1 daniel suporte 136 Oct  2 00:49 agenda
daniel@casadocodigo:~$
```

Ao alterar o grupo, agora estamos dando permissão para que usuários que pertencem ao grupo `suporte` possam ter acesso ao arquivo `agenda`, neste caso o usuário `paulo`.

O que aconteceria se alterássemos o dono do arquivo? Para fazer isso utilizamos o comando `chown`:

```
daniel@casadocodigo:~$ sudo chown paulo:suporte agenda
daniel@casadocodigo:~$ ls -l agenda
-rw-rw-r-- 1 paulo suporte 136 Oct  2 00:49 agenda
daniel@casadocodigo:~$
```

Agora o dono do arquivo é o usuário `paulo`. Passamos como argumento o nome do grupo separado por `:` para o comando `chown`. Esta é a sintaxe do comando para alterar o dono e o grupo. Para ver mais opções, consulte a documentação.

Agora que mudamos o dono e o grupo do arquivo, será que o nosso usuário ainda possui permissão para fazer alterações? Tente editar o arquivo e veja o que acontece:

```
Paulo Silveira  @paulo_caelum
Nando Vieira    @fnando
Adriano Almeida @adrianoalmeida7
Vinicius Baggio @vinibaggio
~
~
~
~
~
~
~
~
~
~
~
-- INSERT -- W10: Warning: Changing a readonly file
Press ENTER or type command to continue
```

Figura 6.5: Alerta do editor ao tentar alterar o arquivo

Recebemos uma mensagem de alerta do editor informando que o arquivo está

disponível somente para leitura para o nosso usuário. Para recuperar o acesso ao arquivo, altere o dono e o grupo para o seu usuário:

```
daniel@casadocodigo:~$ sudo chown daniel:daniel agenda
daniel@casadocodigo:~$ ls -l agenda
-rw-rw-r-- 1 daniel daniel 136 Oct 21 01:16 agenda
daniel@casadocodigo:~$
```

Durante todo o processo, note que o `sudo` foi utilizado, pois ele é quem tem o poder de delegar todas essas atribuições no sistema e escolher usuários, grupos e permissões.

Com as permissões corretas, podemos executar aplicações que só determinados usuários podem. Pra executar programas interessantes, vamos aprender a instalá-los!

Capítulo 7

Instalando pacotes e aplicativos

Cada distribuição Linux possui pacotes específicos. Vamos aprender um pouco sobre eles, como instalá-los e removê-los do sistema.

Primeiramente, vamos entender o que é um pacote. Um pacote é um conjunto de arquivos agrupados para facilitar a instalação e distribuição de um programa. Ele pode conter `scripts` para listagem e checagem de dependências para configuração durante o processo de instalação de um aplicativo.

No *Ubuntu* os pacotes são baseados em *Debian* e têm a extensão `.deb`.

7.1 Gerenciador de pacotes

Veremos agora sobre gerenciamento de pacotes para instalação, atualização e remoção de aplicativos. O gerenciador de pacotes trabalha interpretando a necessidade de cada pacote para que ele possa funcionar de forma correta.

Antes de prosseguirmos com nosso estudo sobre shell e aprender sobre gerenciamento de pacotes no terminal, veremos um exemplo prático de instalação de um aplicativo no modo gráfico.

Vamos instalar o `google chrome` usando a interface gráfica de gerenciamento de pacotes. Para isso, faça o download do pacote `.deb` em http://www.google.com/intl/pt-BR/chrome/:

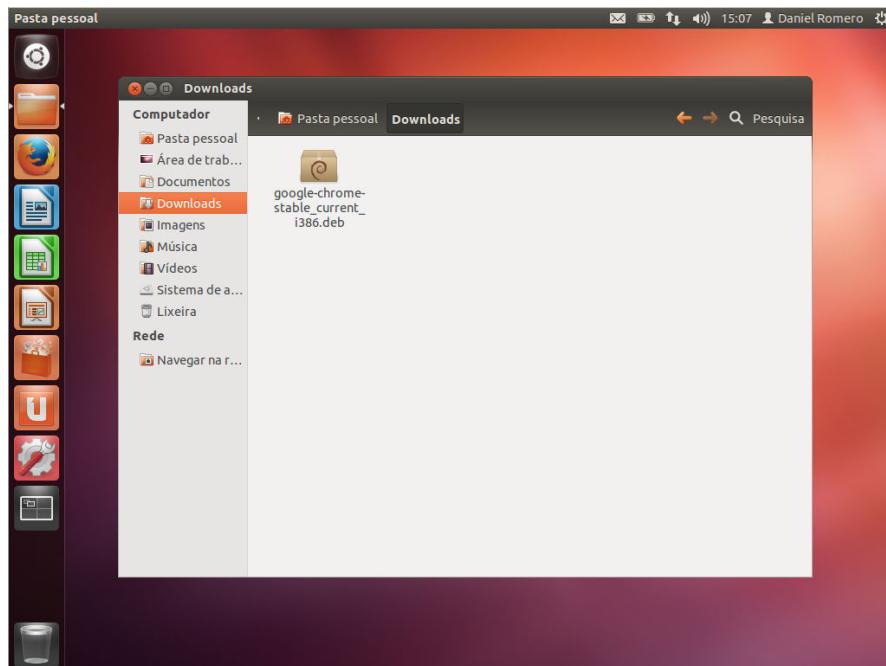



Figura 7.1: Download do pacote .deb do google chrome

De posse do pacote, vamos à instalação. Clique com o botão direito do mouse em cima do pacote e escolha a opção *Abrir com Central de programas do Ubuntu*:

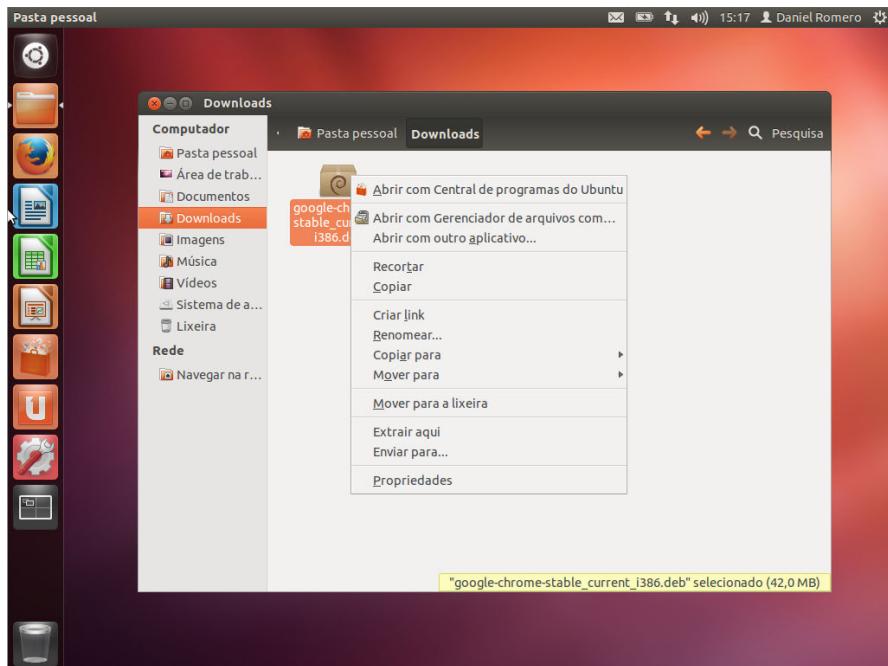



Figura 7.2: Instalando o pacote usando o gerenciador gráfico

Agora, basta clicar em `Instalar` e teremos o `google chrome` instalado em nosso sistema:

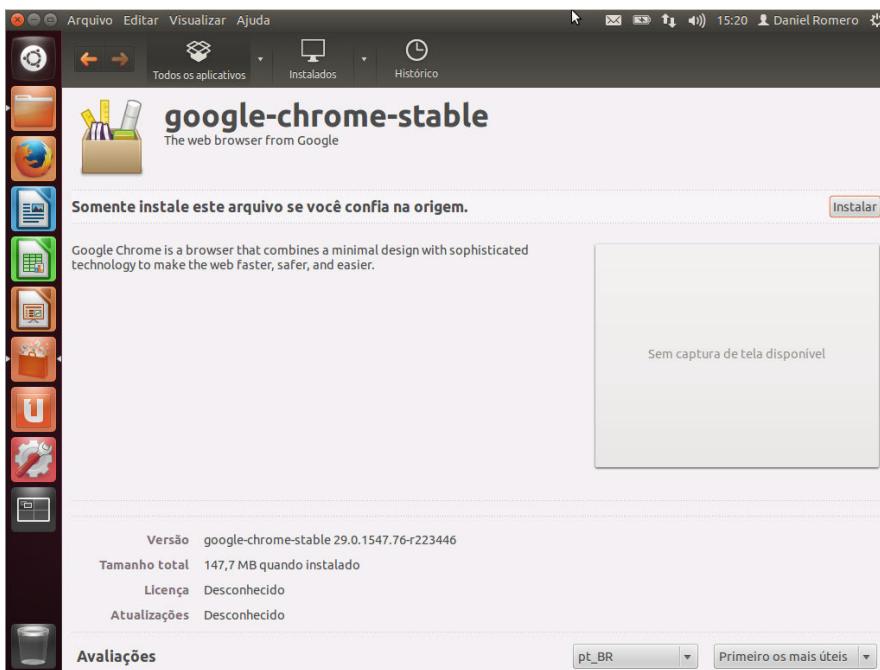



Figura 7.3: Efetivando a instalação

O gerenciador irá solicitar a senha do seu usuário, que é a mesma que você cadastrou durante o processo de instalação. Informe a senha e clique em `Autenticar`:

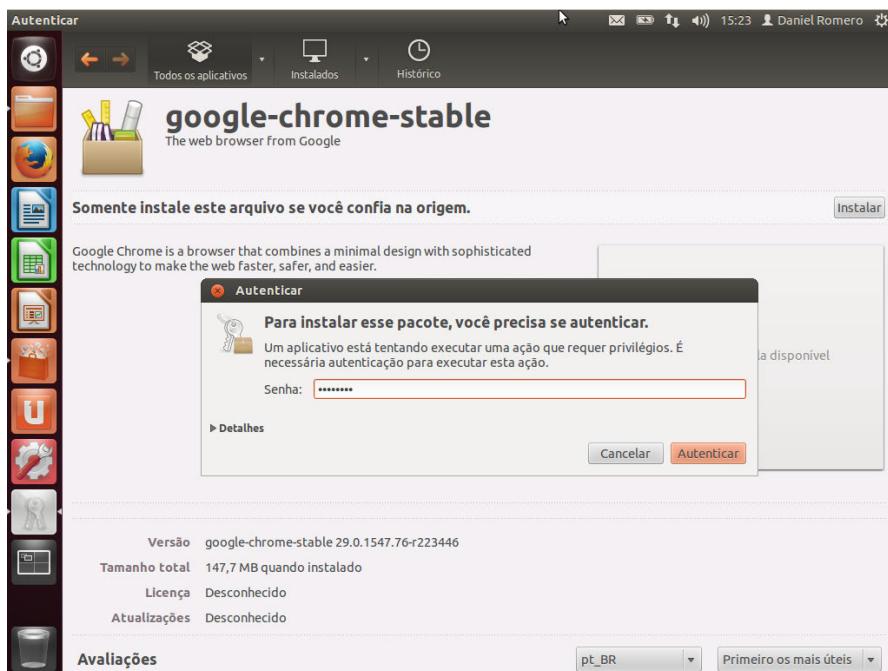



Figura 7.4: Autenticação necessária para instalar pacotes

Aguarde até o processo de instalação ser concluído – o gerenciador vai informar que o pacote foi instalado. Teste o novo aplicativo e verifique que tudo ocorreu com sucesso:

Figura 7.5: Testando o google chrome após a instalação

Agora que já vimos uma demonstração de instalação no ambiente gráfico, voltaremos para o **shell** e aprenderemos sobre o gerenciador `apt`.

7.2 Gerenciando pacotes com APT

Veremos aqui um pouco sobre gerenciamento de pacotes no terminal utilizando a ferramenta `APT` *(Advanced Packaging Tool)*. Para isso, usaremos o comando `apt-get`, que é uma interface para a ferramenta `APT`. Assim como o `apt-get`, existe outra interface para o `APT`, o `aptitude`. Mas em nossos estudos usaremos o `apt-get` para instalar, atualizar e remover aplicativos.

Lembre-se de sempre verificar o manual de um comando. Faça uso excessivo do `man`!

Antes de prosseguir, vamos entender algumas opções do `apt-get`:

- `update`: atualiza a lista de pacotes;
- `upgrade`: atualiza todo o sistema;

- `install`: instala um novo programa;
- `remove`: desinstala um programa.

Para entender melhor como funciona o `apt-get`, vamos fazer a instalação do *Java*. O pacote Oracle JDK foi removido do repositório oficial do Ubuntu devido à nova licença do *Java*. Por conta disso, usaremos um repositório alternativo para instalar o pacote do Java. Veremos mais uma opção do `apt-get`, onde adicionamos um novo canal de softwares usando a opção `add-apt-repository ppa:xxxxxx/xxxx`. Lembre-se de utilizar o `sudo`:

```
daniel@casadocodigo:~$ sudo add-apt-repository ppa:webupd8team/java
```

O `sudo` vai solicitar sua senha, informe-a para continuar e tecle `ENTER` quando for solicitado:

```
daniel@casadocodigo:~$ sudo add-apt-repository ppa:webupd8team/java
[sudo] password for daniel:
You are about to add the following PPA to your system:
 Oracle Java (JDK) Installer (automatically downloads and installs
 Oracle JDK6 / JDK7 / JDK8). There are no actual Java files in this PPA.
  More info: http://www.webupd8.org/2012/01/
 install-oracle-java-jdk-7-in-ubuntu-via.html

Debian installation instructions:
 http://www.webupd8.org/2012/06/
 how-to-install-oracle-java-7-in-debian.html
 More info: https://launchpad.net/~webupd8team/+archive/java
 Press [ENTER] to continue or ctrl-c to cancel adding it

 gpg: keyring `/tmp/tmpXshrbm/secring.gpg' created
 gpg: keyring `/tmp/tmpXshrbm/pubring.gpg' created
 gpg: requesting key EEA14886 from hkp server keyserver.ubuntu.com
 gpg: /tmp/tmpXshrbm/trustdb.gpg: trustdb created
 gpg: key EEA14886: public key "Launchpad VLC" imported
 gpg: Total number processed: 1
 gpg:               imported: 1  (RSA: 1)
 OK
```

Após adicionar a nova fonte de repositório, é necessário atualizar a lista de pacotes – usaremos a opção `update`:

```
daniel@casadocodigo:~$ sudo apt-get update
```

Depois de atualizar a lista de pacotes, vamos finalmente instalar o *Java*, usando a opção `install`:

```
daniel@casadocodigo:~$ sudo apt-get install oracle-java7-installer
```

Ao rodar o comando anterior, o `APT` vai informar o que será instalado junto com as dependências. Quando o `APT` perguntar se deseja continuar, tecle `Y` para informar que sim:

```
daniel@casadocodigo:~$ sudo apt-get install oracle-java7-installer

Reading package lists... Done

Building dependency tree
Reading state information... Done
The following extra packages will be installed:
gsfonts gsfonts-x11 java-common
Suggested packages:
default-jre equivs binfmt-support visualvm ttf-baekmuk
ttf-unfonts ttf-unfonts-core ttf-kochi-gothic
ttf-sazanami-gothic ttf-kochi-mincho ttf-sazanami-mincho
ttf-arphic-uming firefox firefox-2 iceweasel mozilla-firefox
iceape-browser mozilla-browser epiphany-gecko
epiphany-webkit epiphany-browser galeon midbrowser
moblin-web-browser xulrunner xulrunner-1.9 konqueror
chromium-browser midori google-chrome
The following NEW packages will be installed:
gsfonts gsfonts-x11 java-common oracle-java7-installer
0 upgraded, 4 newly installed, 0 to remove and 18 not upgraded.
Need to get 3,462 kB of archives.
After this operation, 5,351 kB of additional disk space will be used.
Do you want to continue [Y/n]? Y
```

O instalador do Java vai informar sobre os termos de licença de uso. Para continuar, tecle `Enter`:

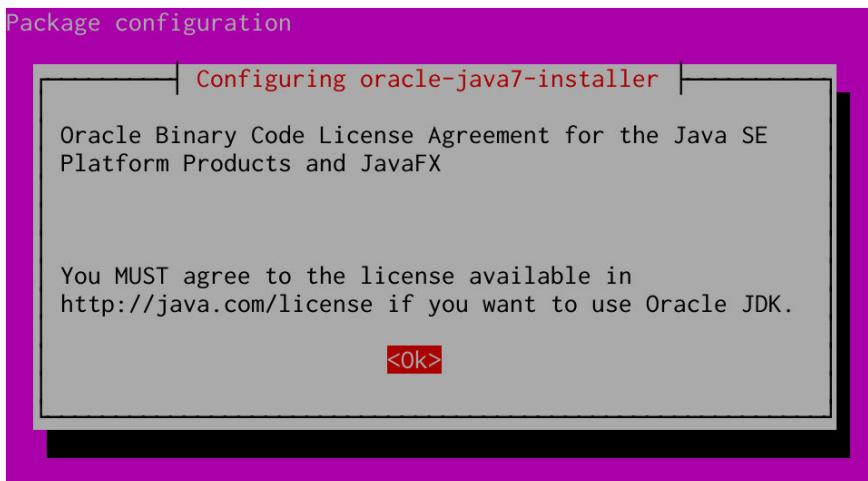



Figura 7.6: Java installer

Em seguida o instalador pergunta se você aceita os termos de uso da licença. Escolha `Yes` e tecle `Enter` para continuar e finalizar a instalação:

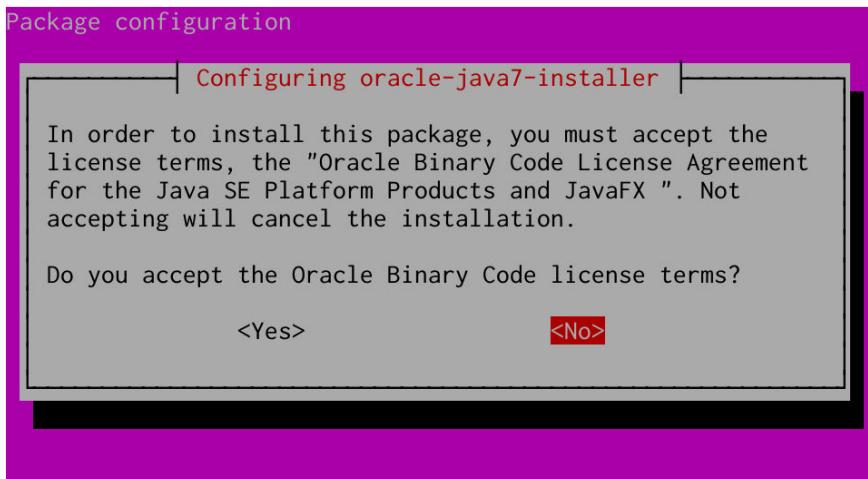



Figura 7.7: Java installer licença de uso

Para finalizar vamos conferir a versão do Java que foi instalada. Execute `java -version`:

```
daniel@casadocodigo:~$ java -version
java version "1.7.0_40"
Java(TM) SE Runtime Environment (build 1.7.0_40-b43)
Java HotSpot(TM) Client VM (build 24.0-b56, mixed mode)
daniel@casadocodigo:~$
```

Pronto! Instalação com sucesso.

Capítulo 8

Prática, instalando Apache, PHP e MySQL

Neste capítulo iremos praticar tudo o que vimos até aqui, desde navegação, edição de arquivos até instalação de pacotes. Conheceremos um pouco sobre configuração e controle de serviços e entenderemos mais sobre usuários e grupos.

Nosso objetivo será de instalar e configurar o *web server Apache*, a linguagem de programação *PHP* e o banco de dados *MySQL*. Faremos os 3 funcionar em conjunto.

Não se preocupe caso você não os conheça. São softwares muito utilizados e foram escolhidos como exemplo para que você entenda bem o mecanismo de instalação de pacotes.

8.1 Instalando o Apache

Vamos começar instalando primeiro o *Apache*. Para isso, antes precisamos atualizar a nossa lista de pacotes e instalar um pacote essencial para o processo de compilação

de outros pacotes.

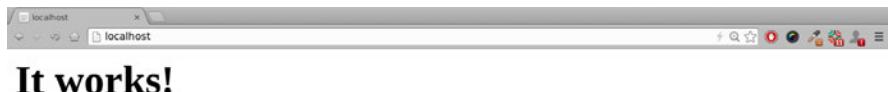
Ele é o `build-essential`, que possui, por exemplo, compilador para a linguagem C (o GCC) e outras ferramentas.

Atualize a lista de pacotes do seu repositório, fazendo o comando que já conhecemos:

```
daniel@casadocodigo:~$ apt-get update
[sudo] password for daniel:
```

Informe a senha do seu usuário e prossiga com o processo de atualização. Em seguida, podemos instalar o pacote `build-essential`:

```
daniel@casadocodigo:~$ sudo apt-get install build-essential
```

Agora instale o pacote `apache2`. Serão listadas as dependências que serão instaladas junto ao pacote `apache2` – o sistema deve solicitar a confirmação para prosseguir com a instalação. Faça:

```
daniel@casadocodigo:~$ sudo apt-get install apache2
```

Sempre que quiser obter informações sobre um pacote, utilize o comando `apt-cache`, que possui a seguinte sintaxe `apt-cache show pacote`. Faça um teste com o pacote `apache2`:

```
daniel@casadocodigo:~$ sudo apt-cache show apache2
```

Podemos notar que foram informados muitos detalhes sobre o pacote, como lista de dependências, versão e descrição.

Se o *Apache* foi instalado com sucesso, já deve estar funcionando. Faça um teste acessando pelo browser o endereço http://localhost ou http://127.0.0.1, e você verá uma mensagem do *Apache* informando que o web server está funcionando:

Figura 8.1: Apache instalado e funcionando

Agora que temos o web server *Apache* instalado e funcionando, podemos fazer testes para entender sobre serviços. Após a instalação, o web server fica disponível no sistema em forma de serviço; sendo assim, usaremos o comando `service` para verificar as opções disponíveis para o serviço `apache2` que instalamos:

```
daniel@casadocodigo:~$ sudo service apache2
 * Usage: /etc/init.d/apache2 {start|stop|graceful-stop|restart|reload|
 force-reload|start-htcacheclean|stop-htcacheclean|status}
 daniel@casadocodigo:~$
```

Note que o comando `service` executa o script de serviços `apache2` localizado em `/etc/init.d/apache2`, que possui as várias opções como iniciar, parar, restartar e outras.

Teste o `service` parando o serviço `apache2`. Para isso, execute:

```
daniel@casadocodigo:~$ sudo service apache2 stop
 * Stopping web server apache2
apache2: Could not reliably determine the server's fully qualified domain
  name, using 127.0.0.1 for ServerName
```

```
... waiting .                                                    [ OK ]
daniel@casadocodigo:~$
```

Foi informado que o o serviço `apache2` foi interrompido. Vá ao browser e dê refresh para verificar que a página *It works!* não está mais acessível. Para iniciar o serviço novamente, faça:

```
daniel@casadocodigo:~$ sudo service apache2 start
 * Starting web server apache2
apache2: Could not reliably determine the server's fully qualified domain
name, using 127.0.0.1 for ServerName                                  [ OK ]
daniel@casadocodigo:~$ ]
```

Usaremos bastante o comando `service` daqui pra frente, lembre-se de pesquisar sobre ele com o `man`.

Reinicie sua máquina e veja que, por padrão, o serviço do Apache é inicializado automaticamente. Alguns pacotes, como o Apache, além de instalar um serviço, torna-o autoinicializável.

8.2 Linkando arquivos

Antes de prosseguir faremos um pequeno desvio para ver um pouco sobre linkagem de arquivos, ou seja, como fazer um *link*. É algo que aparece com frequência no Linux.

O `apache` faz bastante uso de linkagem de arquivos em suas configurações, assim, agora temos exemplos mais ricos.

Para fazer um link, utilizamos o comando `ln -s`, que cria um link simbólico para o arquivo de origem – seu conteúdo é o caminho do arquivo. Para entender melhor, observe a listagem nas configurações do `apache`:

```
daniel@casadocodigo:/etc/apache2/sites-enabled$ ls -l
total 0
lrwxrwxrwx 1 root root 26 Oct  3 23:11 000-default ->
../sites-available/default
daniel@casadocodigo:/etc/apache2/sites-enabled$
```

Note que o arquivo `000-default` está apontando para o arquivo `default`, que fica no diretório `sites-available`. Vamos criar um link simbólico para o default com outro nome, para isso devemos antes remover o link que já existe:

```
daniel@casadocodigo:/etc/apache2/sites-enabled$ sudo rm 000-default
daniel@casadocodigo:/etc/apache2/sites-enabled$ sudo ln -s
../sites-available/default daniel-config
daniel@casadocodigo:/etc/apache2/sites-enabled$
daniel@casadocodigo:/etc/apache2/sites-enabled$ ls -l
total 0
lrwxrwxrwx 1 root root 26 Oct 20 21:23 daniel-config ->
../sites-available/default
daniel@casadocodigo:/etc/apache2/sites-enabled$
```

É importante saber que existem dois tipos de ligações:

- **Hard link**: cria um arquivo igual capaz de compartilhar os dados;
- **Link simbólico**: é somente um caminho para o arquivo original.

No exemplo anterior, criamos um link simbólico. Sem a opção `-s` no comando `ln`, teríamos criado um hard link. Agora qualquer alteração no link `daniel-config` que se encontra no diretório `sites-enabled` será refletida no arquivo `default`, que fica em `sites-available`.

O Linux utiliza bastante o recurso de linkagem de arquivos, até mesmo para representar o hardware durante o processo de comunicação com o kernel. Liste de forma detalhada o diretório `/dev` para ver a quantidade de *hard links* e *links simbólicos* utilizados pelo sistema.

Agora que entendemos um pouco deste recurso, vamos prosseguir!

8.3 Instalando e configurando o MySQL

Vamos prosseguir instalando e configurando o *MySQL*. Ele é um sistema de gerenciamento de banco de dados relacional bastante popular, muito usado em conjunto com aplicações bem conhecidas como o *WordPress*.

O primeiro passo é instalar o pacote `mysql-server`, e para isso faça:

```
daniel@casadocodigo:~$ sudo apt-get install mysql-server
```

Mais uma vez, o `apt-get` vai informar os pacotes extras que serão instalados e o total de espaço em disco que será consumido na instalação. Sua confirmação é necessária para continuar.

Durante o processo de instalação do `mysql` será solicitada a definição de senha para o usuário `root`:

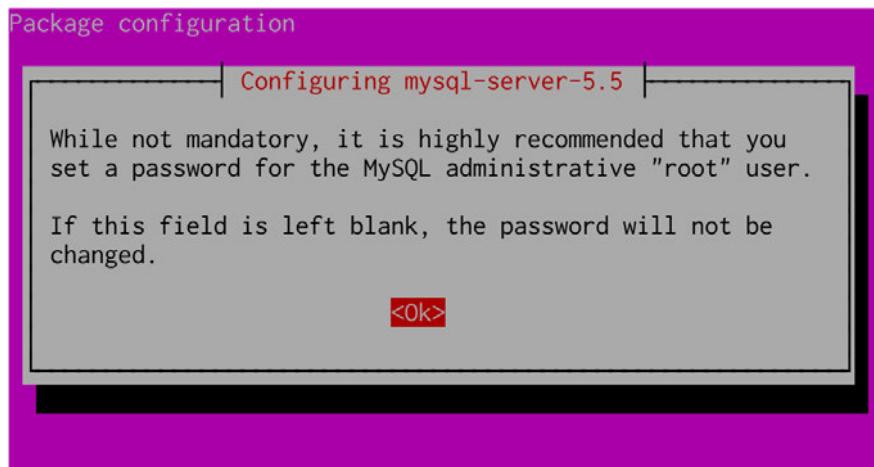



Figura 8.2: Instalação do MySQL

Defina a senha de usuário `root` ou simplesmente tecle `Enter` pois é opcional, já que se trata de uma instalação em sua máquina local:

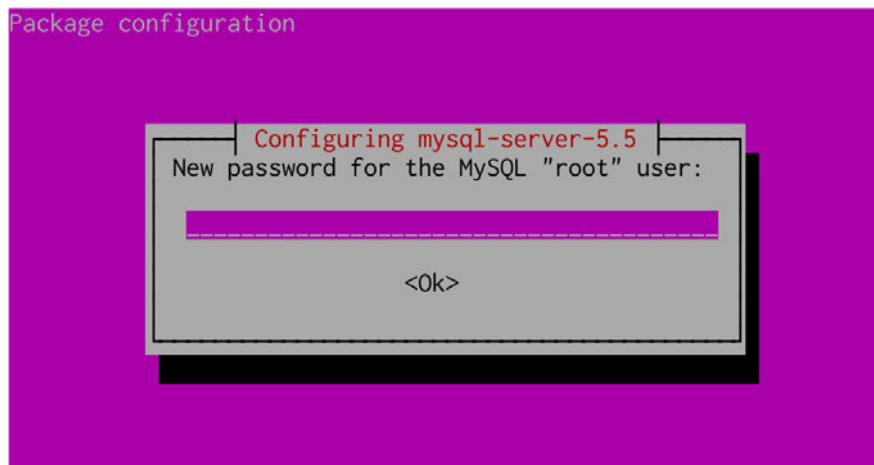



Figura 8.3: Definição de senha para o usuário root do MySQL

Após a instalação, verifique se o serviço `mysql` está rodando em seu sistema. Use o comando `service` com a opção `status`:

```
daniel@casadocodigo:~$ sudo service mysql status
mysql start/running, process 11879
daniel@casadocodigo:~$
```

Note que ele informa que o serviço está rodando. A informação `process 11879` exibida refere-se ao número do processo do serviço no sistema. Esse número vai variar, e será provavelmente diferente na sua máquina. Não se preocupe, entenderemos sobre processos no próximo capítulo.

Agora teste o `mysql` efetuando login no serviço com o usuário `root`:

```
daniel@casadocodigo:~$ sudo mysql -u root -p
Enter password:
```

O `mysql` solicitará a senha de `root` que foi definida no processo de instalação. Após informá-la, você verá uma tela como a seguinte:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 38
Server version: 5.5.32-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All righ
ts reserved.

Oracle is a registered trademark of Oracle Corporation and/or it
s
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current in
put statement.

mysql>
```

Figura 8.4: Conectando ao MySQL pelo shell

Liste os bancos de dados disponíveis:

```
mysql> show databases;
+--------------------+
```

```
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.00 sec)

mysql>
```

Para sair, faça:

```
mysql> quit
Bye
daniel@casadocodigo:~$
```

Assim como o `apache`, o `mysql` também é um serviço e, como tal, possui as mesmas regras disponíveis no `service`. Isso possibilita parar, iniciar, recarregar, ver o status, igual fizemos no `apache`:

```
daniel@casadocodigo:~$ sudo service mysql stop
mysql stop/waiting
daniel@casadocodigo:~$ sudo service mysql status
mysql stop/waiting
daniel@casadocodigo:~$ sudo service mysql start
mysql start/running, process 12089
daniel@casadocodigo:~$ sudo service mysql status
mysql start/running, process 12089
daniel@casadocodigo:~$
```

8.4 Instalando e configurando o PHP

Vamos prosseguir com a nossa prática instalando e configurando o *PHP*. Instalaremos três pacotes, `php5` – pacote da linguagem, `php-pear` – pacote com classes base para o `php` e o `php5-mysql` – pacote para comunicação com o banco de dados `mysql`. Para instalar faça:

```
daniel@casadocodigo:~$ sudo apt-get install php5 php-pear php5-mysql
```

Agora que instalamos o `php`, precisamos ajustar o seu arquivo de configuração que está em `/etc/php5/apache2/php.ini`. Vamos permitir mensagens de erro

mais descritivas e ativar logs. Para isso, vamos editar o `php.ini`. Utilize o `vim` ou `nano`:

```
daniel@casadocodigo:~$ sudo vim /etc/php5/apache2/php.ini
```

Com o arquivo aberto, busque por `error_reporting`. No `vim`, faça:

```
/error_reporting =
```

Essa instrução no `vim` irá buscar pelo texto `error_reporting =` no arquivo. Após encontrar, edite a linha e faça a seguinte alteração:

De:

```
error_reporting = E_ALL & ~E_DEPRECATED
```

Para:

```
error_reporting = E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|
E_CORE_ERROR
```

Desta forma, estamos alterando o formato das mensagens de erro do `php` para serem mais descritivas. Agora vamos especificar o arquivo de logs onde o `php` deverá salvar: busque no `vim` a diretiva `error_log =` e descomente apagando o `;` no inicio da linha. Em seguida, insira o seguinte caminho:

```
error_log = /var/log/php.log
```

Pronto, agora os logs do `php` irão ser salvos no arquivo `php.log` localizado no diretório `/var/log`. Salve o arquivo, saia do editor e no `vim` faça: `:wq`.

De volta ao `shell`, agora que alteramos as configurações do `php` precisamos recarregar o `apache` para que ele entenda as novas configurações setadas no `php`. Execute:

```
daniel@casadocodigo:~$ sudo service apache2 reload
```

A opção `reload` do `apache` vai recarregar os módulos e as novas configurações sem precisar restartar o serviço por completo.

Agora que instalamos e configuramos o `php`, é uma boa ideia verificar se está funcionando perfeitamente. Para isso, vamos criar um script de teste em `php` e conferir se está tudo ok. Crie um novo arquivo chamado de `teste.php` em `/var/www`, e com o `vim` faça:

```
daniel@casadocodigo:~$ sudo vim /var/www/teste.php
```

Insira no arquivo o seguinte trecho de código:

```php
<?php
  phpinfo();
?>
```

Salve o arquivo e saia do editor com `:wq`; em seguida confira o resultado no browser, acessando http://localhost/teste.php.

PHP Version 5.3.10-1ubuntu3.8

System	Linux precise32 3.2.0-23-generic-pae #36-Ubuntu SMP Tue Apr 10 22:19:09 UTC 2012 i686
Build Date	Sep 4 2013 19:47:29
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/pdo.ini
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,NTS
PHP Extension Build	API20090626,NTS
Debug Build	no
Thread Safety	disabled

Figura 8.5: Teste para verificar se o php foi instalado corretamente

Quer um desafio? Instale o WordPress. Existem várias formas de fazer isso. Talvez a mais simples será baixar os arquivos fontes do http://www.wordpress.org e descompactá-los em um diretório. Após isso, basta configurar o Apache, editando seu arquivo de configuração, para que esse diretório seja visualizado por um *virtual host*. Sim, você terá de ler algumas páginas de instruções, mas você já possui todo o conhecimento de Linux básico para tal!

Caso se interesse por programação, não deixe de conhecer os livros dos meus outros colegas da Casa do Código, inclusive sobre PHP.

Capítulo 9

Entendendo processos

Vamos nos aprofundar um pouco mais no Linux e conhecer sobre o seu background. Veremos como funciona sua inicialização e como os serviços são gerenciados, mas antes precisamos entender o que são processos.

9.1 O que são processos?

A definição de processo apresentada por Tanenbaum no livro **Sistemas Operacionais - Projeto e Implementação**:

A ideia-chave aqui é que um processo é um tipo de atividade. Ele tem um programa, entrada, saída e um estado. Um único processador pode ser compartilhado entre vários processos, com algum algoritmo de agendamento sendo utilizado para determinar quando parar de trabalhar em um processo e servir a um diferente.

Pense em um processo como a representação de um programa em execução utilizando os recursos do computador para realizar alguma tarefa.

Um processo possui estados que definem o seu comportamento, são eles:

- `execução`: o processo está ativo utilizando a CPU e outros recursos;
- `pronto` ou `espera`: o processo está temporariamente parado permitindo que outro processo execute na sua frente;
- `bloqueado`: o processo está parado aguardando a execução de algum evento para voltar ao estado de `execução`.

Além de possuir esses comportamentos, um processo é capaz de criar outros processos. Quando isso ocorre dizemos que um processo é pai dos outros criados por ele.

Cada processo no Linux recebe um número para sua identificação conhecido por `PID`. Podemos vê-los com o comando `ps`:

```
daniel@casadocodigo:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY    STAT START   TIME COMMAND
root         1  0.5  0.3   3516  1908 ?      Ss   18:49   0:00 /sbin/init
root         2  0.0  0.0      0     0 ?      S    18:49   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?      S    18:49   0:00 [ksoftirq]
root         4  0.0  0.0      0     0 ?      S    18:49   0:00 [kworker/]
root         5  0.1  0.0      0     0 ?      S    18:49   0:00 [kworker/]
```

Certamente, no seu computador, o resultado será diferente, apresentando outros processos, com outros ids.

9.2 O processo init

Quando inicializamos o Linux, o primeiro processo criado é o `init`, que é conhecido como o pai de todos os outros.

Depois de inicializar o ambiente gráfico, quando abrimos o terminal, um processo é criado para controlar o terminal em questão. De forma semelhante, para cada programa aberto também será criado um processo correspondente. O `init` é responsável por inicializar todos eles e possui a identificação de número 1 no sistema.

9.3 A identificação de processos

Como vimos, um processo é identificado por seu `PID` (Process Identifier). Esse número é dado pelo sistema para cada processo, cada `PID` é único, então você nunca verá dois ou mais processos fazendo uso do mesmo `PID`.

Cada processo também possui um usuário dono, dessa forma, o sistema verifica as permissões e sabe qual usuário pode executar um determinado processo. A identificação de donos é feita pelos números `UID` e `GID`.

No Linux, todo usuário possui um número de identificação da mesma forma que os processos. Esse número é conhecido por `UID` (User Identifier) e o `GID` (Group Identifier).

9.4 Verificando processos

Verificar e gerenciar processos é uma tarefa muito importante, pois às vezes precisamos interromper um processo à força ou verificar quais processos estão consumindo mais recursos no computador (CPU, memória etc).

Veremos alguns comandos novos. Vamos começar pelo comando `ps`, que serve para listar os processos em execução e obter informações como `PID` e `UID`.

Execute o `ps` para obter a lista de processos do nosso usuário:

```
daniel@casadocodigo:~$ ps
  PID TTY          TIME CMD
 1788 pts/0    00:00:00 bash
 1847 pts/0    00:00:00 ps
daniel@casadocodigo:~$
```

No caso, o usuário `daniel` possui apenas 2 processos em execução: o `bash`, que é o processo do `shell`, e o próprio `ps`, que foi o processo criado ao executar o comando `ps`.

Assim como todos os comandos que vimos até agora, o `ps` também possui opções. Veremos algumas delas:

- `a`: lista todos os processos existentes;
- `u`: exibe o nome do usuário dono do processo;
- `x`: lista os processos que não possuem relação com o terminal;
- `m`: exibe a quantidade memória consumida por cada processo.

Para mais opções `man ps`, a combinação mais usada é `ps aux` – execute e veja a lista de processos em execução:

```
daniel@casadocodigo:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY   STAT START   TIME COMMAND
root         1  0.0  0.3   3516  1908 ?     Ss   18:49   0:00 /sbin/init
root         2  0.0  0.0      0     0 ?     S    18:49   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?     S    18:49   0:00 [ksoftirq]
root         5  0.0  0.0      0     0 ?     S    18:49   0:00 [kworker/]
root         6  0.0  0.0      0     0 ?     S    18:49   0:00 [migratio]
root         7  0.0  0.0      0     0 ?     S    18:49   0:00 [watchdog]
root         8  0.0  0.0      0     0 ?     S<   18:49   0:00 [cpuset]
root         9  0.0  0.0      0     0 ?     S<   18:49   0:00 [khelper]
root        10  0.0  0.0      0     0 ?     S    18:49   0:00 [kdevtmpf]
```

Note que podemos ver o usuário dono de cada processo na primeira coluna `USER`. Em seguida temos o `PID` de cada processo, informações de consumo de `CPU` e memória, outras informações como data e hora de quando o processo foi inicializado, assim como o nome do processo.

Podemos contar todos os processos em execução fazendo uso de um comando que ainda não testamos: o `wc`. Basicamente o que ele faz é contar as linhas de um arquivo ou do conteúdo que for exibido no terminal, para isso utilizamos a opção `wc -l`. Para informar ao comando que queremos apenas a quantidade de linhas, o que faremos é executar o `ps aux` em combinação com o `wc -l`, usando o `|` (pipe), que é uma forma de encadeamento de processos:

```
daniel@casadocodigo:~$ ps aux | wc -l
82
daniel@casadocodigo:~$
```

O que o `|` fez foi encadear a execução do comando `ps aux` ao comando `wc -l`. Assim, ele pegou o resultado gerado do primeiro comando e passou para o segundo. A saída do `wc` nos mostra quantos processos ao todo temos rodando em nosso computador.

Outro comando que ainda não vimos é o `grep`, que procura por uma expressão que pode ser uma palavra ou frase em um arquivo, ou ainda pode funcionar como filtro na saída de comandos. Vamos usá-lo para filtrar a lista de processos gerada com `ps -A` e buscar todos os processos do `apache` que estiverem em execução. A opção `-A` do comando `ps` é para exibir todos os processos em execução mas sem detalhes.

```
daniel@casadocodigo:~$ ps -A | grep apache
 1166 ?        00:00:00 apache2
```

```
 1184 ?        00:00:00 apache2
 1185 ?        00:00:00 apache2
 1186 ?        00:00:00 apache2
 1187 ?        00:00:00 apache2
 1188 ?        00:00:00 apache2
daniel@casadocodigo:~$
```

Foram retornados somente os processos com nome `apache`, pois o `grep` realizou um filtro para ignorar todo o resto e exibir o que procurávamos. Experimente executar `ps -A` para ver o tamanho da lista de processos.

Veremos agora outro comando bastante usado para verificar processos, o `top`, que acompanha os processos atualizando as informações quase em tempo real. Execute no terminal o `top` para vê-lo funcionando:

```
top - 21:26:16 up  2:37,  1 user,  load average: 0.07, 0.03, 0.05
Tasks:  80 total,   1 running,  79 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.
Mem:    507536k total,   208692k used,   298844k free,    28164k buffers
Swap:   786428k total,        0k used,   786428k free,   110176k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2045 daniel    20   0  2720 1104  880 R  0.3  0.2   0:00.17 top
    1 root      20   0  3516 1908 1304 S  0.0  0.4   0:00.94 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.04 ksoftirqd/0
    5 root      20   0     0    0    0 S  0.0  0.0   0:00.32 kworker/u:0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:00.09 watchdog/0
    8 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 cpuset
    9 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 khelper
   10 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kdevtmpfs
   11 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 netns
   12 root      20   0     0    0    0 S  0.0  0.0   0:00.04 sync_supers
   13 root      20   0     0    0    0 S  0.0  0.0   0:00.00 bdi-default
   14 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kintegrityd
   15 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kblockd
```

Figura 9.1: Executando o top

Na parte superior, logo nas primeiras linhas, temos informações sobre o sistema, com número total de processos, uso da `CPU`, uso da memória. Em seguida, temos a lista dos processos existentes. Para obter ajuda sobre o uso, tecle `h` e, para sair do `top`, tecle `q`. O top possui muitas opções, veja algumas em `man top`.

Uma opção interessante é acompanhar os processos de um determinado usuário do sistema. Para isso usamos a opção `-u` e o nome do usuário:

```
daniel@casadocodigo:~$ top -u daniel
```

```
top - 21:37:05 up  2:47,  1 user,  load average: 0.05, 0.05, 0.05
Tasks:  79 total,   1 running,  78 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  0.0%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.
Mem:    507536k total,   208700k used,   298836k free,    28164k buffers
Swap:   786428k total,        0k used,   786428k free,   110176k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1787 daniel    20   0  5316 1328 1064 S  0.0  0.3   0:00.03 su
 1788 daniel    20   0  7820 3968 1592 S  0.0  0.8   0:00.20 bash
 2053 daniel    20   0  2720 1116  888 R  0.0  0.2   0:00.00 top
```

Figura 9.2: Executando filtro no top

Dessa forma, estamos executando um filtro para exibir somente os processos pertencentes ao usuário `daniel`.

Uma outra opção ao `top` é o `htop`, que tem uma interface mais amigável. Ele não vem instalado por padrão, então vamos instalar para conhecê-lo:

```
daniel@casadocodigo:~$ sudo apt-get install htop
```

Após a instalação, execute `htop`:

```
CPU[                              0.0%]   Tasks: 39, 27 thr; 1 running
Mem[|||||||||||||            69/495MB]   Load average: 0.01 0.04 0.05
Swp[                          0/767MB]   Uptime: 02:59:27

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2099 daniel     20   0  5572  1772  1288 R  0.0  0.3  0:00.96 htop
 1049 mysql      20   0  318M 33676  5884 S  0.0  6.6  0:03.00 /usr/sbin/mysqld
 1497 ntp        20   0  5728  2012  1568 S  0.0  0.4  0:01.41 /usr/sbin/ntpd -p
    1 root       20   0  3516  1908  1304 S  0.0  0.4  0:00.94 /sbin/init
  309 root       20   0  2832   608   452 S  0.0  0.1  0:00.07 upstart-udev-bridg
  311 root       20   0  2972  1148   792 S  0.0  0.2  0:00.05 /sbin/udevd --daem
  399 messagebu  20   0  3256   648   448 S  0.0  0.1  0:00.01 dbus-daemon --syst
  417 syslog     20   0 30036  1368  1064 S  0.0  0.3  0:00.02 rsyslogd -c5
  418 syslog     20   0 30036  1368  1064 S  0.0  0.3  0:00.00 rsyslogd -c5
  419 syslog     20   0 30036  1368  1064 S  0.0  0.3  0:00.00 rsyslogd -c5
  414 syslog     20   0 30036  1368  1064 S  0.0  0.3  0:00.05 rsyslogd -c5
  416 root       20   0  2908   392   212 S  0.0  0.1  0:00.00 rpc.idmapd
  464 root       20   0  2968   740   376 S  0.0  0.1  0:00.00 /sbin/udevd --daem
  491 root       20   0  2968   712   348 S  0.0  0.1  0:00.00 /sbin/udevd --daem
  553 root       20   0  2696   944   708 S  0.0  0.2  0:00.01 rpcbind -w
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Figura 9.3: Executando o htop

Bem melhor, não é mesmo?

Agora é possível identificar rapidamente as informações. O consumo de CPU e memória ficou mais amigável, e na barra inferior existe um menu com opções. Por exemplo, tecle `F1` para obter ajuda:

```
htop 1.0.1 - (C) 2004-2011 Hisham Muhammad
Released under the GNU GPL. See 'man' page for more info.

CPU usage bar: [low-priority/normal/kernel/virtualiz                used%]
Memory bar:    [used/buffers/cache                            used/total]
Swap bar:      [used                                          used/total]
Type and layout of header meters are configurable in the setup screen.

 Status: R: running; S: sleeping; T: traced/stopped; Z: zombie; D: disk sl
 Arrows: scroll process list             F5 t: tree view
 Digits: incremental PID search             u: show processes of a single
   F3 /: incremental name search            H: hide/show user threads
   F4 \: incremental name filtering         K: hide/show kernel threads
  Space: tag processes                      F: cursor follows process
      U: untag all processes              + -: expand/collapse tree
   F9 k: kill process/tagged processes      P: sort by CPU%
   ] F7: higher priority (root only)        M: sort by MEM%
   [ F8: lower priority (+ nice)            T: sort by TIME
                                         F4 I: invert sort order
   F2 S: setup                           F6 >: select sort column
 ? F1 h: show this help screen              l: list open files with lsof
  F10 q: quit                               s: trace syscalls with strace
```

Figura 9.4: Obtendo ajuda no htop

Navegue um pouco no `htop`, leia a documentação, para sair tecle `q` ou `F10`.

9.5 O que são sinais de processos?

No Linux os sinais são uma forma de comunicação usada pelos processos para que o sistema consiga interferir em seu funcionamento. Na prática, ao receber um sinal com instruções, um processo interpreta a ação que foi especificada no sinal e a executa.

Alguns dos sinais mais conhecidos e usados por processos são:

- `KILL`: sinal com função de encerrar um processo;
- `TERM`: termina o processo após ele finalizar uma tarefa;
- `STOP`: interrompe a execução de um processo;
- `CONT`: ativa a execução de um processo que foi interrompido.

Para entender melhor sobre sinais, vamos ver um pouco do comando `kill`. Este comando é usado para o envio de sinais a processos, sua sintaxe é simples e depende apenas do `PID` de um processo.

Vamos interromper a execução do `mysql` enviando um sinal de `STOP` para o seu processo. Precisamos primeiro do número `PID` do `mysql`:

```
daniel@casadocodigo:~$ ps -A | grep mysql
 1047 ?        00:00:00 mysqld
daniel@casadocodigo:~$
```

Agora que temos o número `PID` do processo do `mysql`, vamos enviar o sinal de `STOP` para ele, interrompendo sua execução:

```
daniel@casadocodigo:~$ sudo kill -STOP 1047
daniel@casadocodigo:~$
```

Para entender o que realmente ocorreu, tente conectar ao serviço do `mysql`:

```
daniel@casadocodigo:~$ sudo mysql -u root -p
Enter password:
```

Nada vai acontecer, ficaremos o dia inteiro olhando esta tela pois o processo de execução do `mysql` foi interrompido. Abra um novo terminal, envie o sinal `CONT` para ativar a execução do processo que foi interrompido e veja o que acontece no terminal onde estamos tentando nos conectar ao `mysql`:

```
daniel@casadocodigo:~$ sudo kill -CONT 1047
daniel@casadocodigo:~$
```

Ao executar o envio do sinal `CONT` para o processo interrompido do `mysql`, veja que ele voltou a funcionar normalmente.

O comando `kill` também pode ser usado com o número do sinal em vez do seu nome, por exemplo, `-9` representa o sinal `KILL`. Para entender melhor o que o sinal `KILL` faz, abra o editor `vim` em um terminal e em outro terminal busque pelo `PID` do editor. Em seguida, envie o sinal `-9` para o processo:

```
daniel@casadocodigo:~$ ps -A | grep vim
 2488 pts/0    00:00:00 vim
daniel@casadocodigo:~$ sudo kill -9 2488
 [1]+  Killed                  vim
daniel@casadocodigo:~$
```

Ao enviar o sinal `KILL` para o processo do `vim`, ele é encerrado. Recebemos uma mensagem informando que o processo foi *morto*, ou seja, a execução do aplicativo foi finalizada à força.

Outro comando bastante usado é o `killall`, quando não sabemos o `PID` ou quando temos vários processos do mesmo aplicativo com vários `PIDs` em execução, por exemplo o `apache`:

```
daniel@casadocodigo:~$ ps -A | grep apache
 2548 ?        00:00:00 apache2
 2553 ?        00:00:00 apache2
 2554 ?        00:00:00 apache2
 2555 ?        00:00:00 apache2
 2556 ?        00:00:00 apache2
 2557 ?        00:00:00 apache2
daniel@casadocodigo:~$
```

Note que temos vários `PIDs` e, por isso, enviar um sinal de cada vez para cada `PID` torna-se complicado. Neste caso usamos o `killall` e o nome do processo em vez do `PID`. Para interromper a execução do processo, faríamos:

```
daniel@casadocodigo:~$ sudo killall -STOP apache2
daniel@casadocodigo:~$
```

Verifique acessando o endereço http://localhost e note que a página inicial do `apache` não irá carregar. Envie o sinal `CONT` para retornar à execução normal do processo:

```
daniel@casadocodigo:~$ sudo killall -CONT apache2
daniel@casadocodigo:~$
```

Leia mais sobre o `kill` e o `killall` em suas documentações – esses comandos são importantes e bastante utilizados.

Os estados de um processo

Basicamente existem 4 estados para um processo. Após sua criação, o seu estado corrente é `executável`; quando um processo está aguardando alguma rotina para ser executado, dizemos que ele está dormindo – esse estado é chamado de `dormente` –; se um processo está congelado e por algum motivo não pode ser executado, dizemos que seu estado é `parado`; se um processo é considerado *morto*, ou seja, foi finalizado, não está mais em execução mas por algum motivo ainda existe, dizemos que seu estado é de um processo `zumbi`.

9.6 Processos e suas prioridades

Durante sua execução, um processo pode ter prioridade em relação aos outros. Para entender melhor como funcionam as prioridades, vamos ver o conceito de gentileza.

Imagine um processo em execução sendo gentil (oferecendo a gentileza) de deixar um processo com prioridade maior passar a sua frente e ser executado antes. Os processos trabalham com níveis de gentileza, que podem ser definidos através do comando `nice` e um número entre `-19 e 19`, que determina o quão gentil um processo pode ser. Quanto maior for o número definido, mais gentil o processo será, logo quanto menor for o número, maior a sua prioridade.

Normalmente não precisamos determinar as prioridades de um processo pois o Linux trabalha de forma inteligente para fazer isso.

Em alguns casos, por exemplo tarefas de backup, setamos a prioridade para que o processo não consuma de forma inesperada recursos do computador como memória e CPU. Nesses casos, podemos usar 2 comandos: o `nice` e o `renice`.

Por enquanto, vamos testar o `renice` e alterar a prioridade do processo que está executando o `mysql`:

```
daniel@casadocodigo:~$ ps -A | grep mysql
 2340 ?        00:00:03 mysqld
daniel@casadocodigo:~$
```

Agora que temos o `PID` do processo podemos alterar sua prioridade, para isso faça:

```
daniel@casadocodigo:~$ sudo renice -19 2340
2340 (process ID) old priority 0, new priority -19
daniel@casadocodigo:~$
```

Note que o comando nos informa a prioridade antiga que era 0 para a nova prioridade que é -19. Processos com prioridade 0 são intermediários pois estão bem no meio dos extremos. Ao setarmos a prioridade -19, estamos informando que esse processo é pouco gentil e todos os outros deixá-lo-ão passar na frente.

Vamos alterar novamente setando uma nova prioridade e tornando esse processo gentil:

```
daniel@casadocodigo:~$ sudo renice +15 2340
2340 (process ID) old priority -19, new priority 15
daniel@casadocodigo:~$
```

Agora temos um processo gentil de prioridade 15, que irá permitir que outros passem à sua frente e sejam executados antes. É possível ver a prioridade dos processos no `htop`:

```
CPU[                        0.0%]    Tasks: 40, 27 thr; 1 running
Mem[|||||||||||        68/495MB]    Load average: 0.29 0.15 0.11
Swp[                    0/767MB]    Uptime: 04:58:54

 NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  0  318M 33776  5904 S  0.0  6.7  0:00.00 /usr/sbin/mysqld
  0  318M 33776  5904 S  0.0  6.7  0:00.02 /usr/sbin/mysqld
 15  318M 33776  5904 S  0.0  6.7  0:03.66 /usr/sbin/mysqld
  0 34036  6956  3620 S  0.0  1.4  0:00.77 /usr/sbin/apache2 -k s
  0 34108  4264   900 S  0.0  0.8  0:00.00 /usr/sbin/apache2 -k s
  0 34076  4020   664 S  0.0  0.8  0:00.00 /usr/sbin/apache2 -k s
  0 34060  3768   420 S  0.0  0.7  0:00.00 /usr/sbin/apache2 -k s
  0 34060  3768   420 S  0.0  0.7  0:00.00 /usr/sbin/apache2 -k s
  0 34060  3768   420 S  0.0  0.7  0:00.00 /usr/sbin/apache2 -k s
  0 34060  3768   420 S  0.0  0.7  0:00.00 /usr/sbin/apache2 -k s
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F
```

Figura 9.5: Coluna NI mostra a prioridade dos processos

Note, na coluna `NI`, todas as prioridades com 0 e apenas uma opção com valor 15 que é a do processo que foi alterado.

Capítulo 10

Introdução a Shell Script

Script é um programa não compilado que é enviado para um processador. Em seguida, um interpretador lê seu código e o traduz, de modo que o processador entende suas instruções, que serão executadas em seguida.

Shell Script é a linguagem de programação utilizada pelo `shell` – no caso o `shell` é o nosso interpretador que vai ler os nossos programas e dizer ao processador o que fazer.

Programas em *Shell Script* são escritos para tarefas administrativas e repetitivas no Linux. Ao fazer isso, estamos automatizando procedimentos por meio de scripts.

Veremos o básico de Shell Script neste capítulo e, ao final, teremos um script simples para automatizar uma tarefa de backup. Além disso, conheceremos alguns novos comandos bastante utilizados.

10.1 O primeiro script

Vamos ver um exemplo de um programa bem simples:

```
#!/bin/bash
# Nosso primeiro programa em Shell Script

mkdir ~/relatorio
ps aux > ~/relatorio/processos.txt
echo "Programa executado com sucesso!"
```

Nosso primeiro programa é um script muito simples. Linhas que começam com # são comentários ou seja, são desprezadas durante a execução do script. Porém a primeira linha, que até parece um comentário, é na verdade um indicador para informar qual programa irá executar o script. No nosso exemplo, é o programa `bash` que está localizado em `/bin`.

Basicamente, o que o script está fazendo é criar uma pasta chamada `relatorio` no diretório `home` do nosso usuário. Em seguida, é executado o comando `ps aux` para listar todos os processos detalhadamente. A saída desse comando é salvada em um arquivo chamado `processos.txt` dentro da pasta que foi criada. Por fim, o programa emite uma mensagem no terminal informando que foi `"executado com sucesso"`.

10.2 Executando o script

Para executar o nosso primeiro programa é preciso alterar as permissões de forma que o torne um executável para todos os usuários:

```
daniel@casadocodigo:~$ ls -l primeiro_programa.sh
-rw-rw-r-- 1 daniel daniel 147 Oct 10 22:40 primeiro_programa.sh
daniel@casadocodigo:~$ chmod 777 primeiro_programa.sh
daniel@casadocodigo:~$ ls -l primeiro_programa.sh
-rwxrwxrwx 1 daniel daniel 147 Oct 10 22:40 primeiro_programa.sh
daniel@casadocodigo:~$
```

Agora que alteramos a permissão do arquivo `primeiro_programa.sh`, podemos executar da seguinte forma:

```
daniel@casadocodigo:~$ ./primeiro_programa.sh
Programa executado com sucesso!
daniel@casadocodigo:~$
```

Repare no uso do `./`, já que o arquivo está no diretório corrente. Se você tentar executar simplesmente fazendo `primeiro_programa.sh`, verá que o shell não

encontrará o arquivo. Veremos, mais à frente, como modificar o caminho (`PATH`) de procura de arquivos de execução.

Note a mensagem exibida informando que o programa foi executado. Podemos agora verificar se as instruções passadas no script realmente foram executadas. Primeiro veremos se a pasta `relatorio` foi criada no diretório `home` e se o arquivo `processos.txt` foi gerado:

```
daniel@casadocodigo:~$ ls -l ~/relatorio
-rw-rw-r-- 1 daniel daniel 6783 Oct 10 22:56 processos.txt
daniel@casadocodigo:~$
```

Verifique o conteúdo do arquivo `processos.txt` com um dos comandos que já aprendeu, `cat`, `head` ou `tail`. Deve sair algo parecido com:

```
daniel@casadocodigo:~$ head -n 3 relatorio/processos.txt
USER       PID %CPU %MEM    VSZ   RSS TTY    STAT START   TIME COMMAND
root         1  0.0  0.3   3520  1900 ?      Ss   22:34   0:00 /sbin/init
root         2  0.0  0.0      0     0 ?      S    22:34   0:00 [kthreadd]
daniel@casadocodigo:~$
```

Este é o funcionamento de um script: criamos com qualquer editor, salvamos preferencialmente com a extensão `.sh`, alteramos a permissão para que se torne um executável e por fim, rodamos.

Como você já deve ter percebido, basicamente os scripts podem ser instruções com os comandos que você já viu até aqui.

10.3 Operações básicas

Como toda linguagem de programação, Shell Script também trabalha com operações numéricas, variáveis e estruturas de controle. Vamos implementar um exemplo básico contando o número de linhas de um arquivo. Para isso, crie um arquivo chamado `conta_linhas.sh` com o editor de sua escolha e insira o seguinte conteúdo:

```
#!/bin/bash

echo "Contando as linhas ..."
sleep 5
LINHAS=`cat ~/relatorio/processos.txt | wc -l`
echo "Existem $LINHAS no arquivo."
```

Salve o arquivo `conta_linhas.sh`, altere a permissão para que ele torne-se executável com `chmod` e, em seguida, execute o programa com a instrução `./`:

```
daniel@casadocodigo:~$ chmod 777 conta_linhas.sh
daniel@casadocodigo:~$ ./conta_linhas.sh
Contando as linhas ...
Existem 81 no arquivo.
daniel@casadocodigo:~$
```

O que fizemos de diferente?

A instrução `sleep 5` realiza uma contagem de 5 segundos antes de executar as próximas instruções – em outras palavras é um enfeite.

Em seguida criamos uma variável chamada `LINHAS` e atribuímos a ela a instrução para verificar o conteúdo do arquivo `processos.txt` com o comando `cat`, e em seguida contar a quantidade de linhas no arquivo, fazendo uso do comando `wc -l`. Esta atribuição foi feita por meio do sinal `=`. Por fim, é exibida uma mensagem que informa a quantidade de linhas existentes no arquivo.

Esta mensagem possui no seu corpo a chamada da variável que criamos, e para isso utilizamos o `$`.

10.4 Estruturas de controle

Em nossos exemplos, vimos scripts sem opções de controle, ou seja, não ofereciam escolhas, apenas instruções com comandos para execução. Vamos criar um programa simples para verificar e informar se um arquivo existe no diretório `home` do nosso usuário. Para isso, crie um arquivo chamado `verificador.sh`:

```
#!/bin/bash

echo "Informe o nome do arquivo que deseja buscar:"

read ARQUIVO

CONSULTA=$(ls ~ | grep $ARQUIVO)

if [ -z $CONSULTA ]; then
  echo "$ARQUIVO não foi encontrado!"
else
  echo "Arquivo encontrado!"
fi
```

Altere o arquivo dando permissão de execução e em seguida rode o script. Teste buscando por arquivos existentes e não existentes:

```
daniel@casadocodigo:~$ chmod 777 verificador.sh
daniel@casadocodigo:~$ ./verificador.sh
Informe o nome do arquivo que deseja buscar:
Daniel
Daniel não foi encontrado!
daniel@casadocodigo:~$ ./verificador.sh
Informe o nome do arquivo que deseja buscar:
agenda
Arquivo encontrado!
daniel@casadocodigo:~$
```

Note que agora o nosso script ficou mais experto, ele solicita e aguarda que uma instrução seja executada por nós. Neste caso, ele solicita o nome do arquivo que queremos verificar se existe. A instrução `read` é responsável por ler o que vamos digitar no terminal e armazenar para o script continuar com a próxima instrução.

Depois de armazenar a palavra que estamos buscando em uma variável, o script executa uma nova instrução, listando o conteúdo do diretório `home` e filtrando com o comando `grep` em busca da palavra que foi informada.

Finalmente entra a estrutura de controle `if`, e o script verifica se a nossa instrução retorna `nulo` com o parâmetro `-z`. Se a instrução de consulta for nula, será exibida a mensagem informando que o arquivo não foi encontrado, caso contrário, se o arquivo realmente existir, ele será encontrado na busca e a mensagem informará que foi localizado.

No Linux, muitos serviços possuem scripts de controle escritos em Shell Script. Liste o conteúdo do diretório `/etc/init.d` e veja alguns. Note que o `apache` e o `mysql` possuem scripts neste diretório.

A diferenças da listagem para os scripts que fizemos aqui neste capítulo é apenas a falta da extensão `.sh`. Ela não é obrigatória quando definimos na primeira linha do script o programa que irá executá-lo.

Abra com um editor de sua escolha o arquivo `/etc/init.d/apache2` ou utilize o comando `less` para ver o conteúdo do arquivo. Perceba que já nas primeiras linhas vemos algumas instruções que acabamos de aprender:

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          apache2
```

```
# Required-Start:    $local_fs $remote_fs $network $syslog $named
# Required-Stop:     $local_fs $remote_fs $network $syslog $named
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# X-Interactive:     true
# Short-Description: Start/stop apache2 web server
### END INIT INFO

set -e

SCRIPTNAME="${0##*/}"
SCRIPTNAME="${SCRIPTNAME##[KS][0-9][0-9]}"
  if [ -n "$APACHE_CONFDIR" ] ; then
    if ["${APACHE_CONFDIR##/etc/apache2-}" != "${APACHE_CONFDIR}"]; then
      DIR_SUFFIX="${APACHE_CONFDIR##/etc/apache2-}"
    else
      DIR_SUFFIX=
    fi...
```

A declaração do programa que executará o script na primeira linha no exemplo do `apache` é o `/bin/sh`. Em seguida, temos cometários no código, declaração de variáveis e instruções condicionais.

Pode parecer difícil à primeira vista, mas é uma série de comandos condicionais de acordo com variáveis e arquivos de configuração. O Apache é um sistema complexo, há outros arquivos de script bem mais simples. Procure-os entre os diversos serviços já instalados no seu Linux!

O conhecimento de Shell Script é essencial para quem pensa em administrar servidores. Para conhecer mais detalhes desta poderosa linguagem, visite:

http://aurelio.net/shell/

Todo o material produzido por Aurelio Jargas é muito recomendado. Para conhecer mais sobre os operadores e instruções de controle em shell, visite:

http://aurelio.net/shell/canivete/

10.5 Realizando um backup agendado

Antes de prosseguirmos com o Shell Script e implementarmos um script de backup, faremos uma breve pausa para explicar um pouco sobre agendamento de tarefas.

No Linux, o `crontab` é responsável por agendar tarefas para serem executadas em um período de tempo determinado. Ele é gerenciado por um serviço cha-

mado `crond`, que verifica a existencia de agendamentos que devem ser executados. O `crontab` realiza 2 tipos de agendamentos, agendamento dos usuários e agendamento do sistema.

O comando `crontab` pode editar, listar e criar novos agendamentos para usuários, veja as opções em `crontab --help`. Para verificar se existem tarefas agendadas para um determinado usuário faça:

```
daniel@casadocodigo:~$ crontab -u daniel -l
no crontab for daniel
daniel@casadocodigo:~$
```

No exemplo ainda não existe uma tarefa agendada para o usuário `daniel`.

O agendamento dos usuários é armazenado em `/var/spool/cron/crontabs`. Já os agendamentos para o sistema ficam em `/etc`, distribuídos em diretórios que podem ser organizados em agendamentos por hora, dia, semana e mês:

```
daniel@casadocodigo:~$ ls -d /etc/cron.*
/etc/cron.d      /etc/cron.hourly   /etc/cron.weekly
/etc/cron.daily  /etc/cron.monthly
daniel@casadocodigo:~$
```

Vamos dissecar o funcionamento do `crontab` para entender cada parte. Para isso, criaremos um exemplo de uma tarefa agendada para o nosso usuário. A cada minuto será listado o conteúdo da pasta `home` do nosso usuário:

```
daniel@casadocodigo:~$ crontab -u daniel -e
no crontab for daniel - using an empty one

Select an editor.  To change later, run 'select-editor'.
  1. /bin/ed
  2. /bin/nano        <---- easiest
  3. /usr/bin/vim.basic
  4. /usr/bin/vim.tiny

Choose 1-4 [2]:
```

Ao executar o comando `crontab -u daniel -e` para criar uma nova tarefa agendada para o usuário `daniel`, o `crontab` pergunta que editor queremos utilizar. Escolha um de sua preferência – no exemplo, utilizarei o `vim`. Insira e salve no fim do novo arquivo gerado a instrução:

```
*/1 * * * * ls ~ >> ~/crontab.txt
```

O que a instrução faz é listar a cada minuto o conteúdo de `home` do usuário `daniel` e inserir o output da listagem no arquivo `crontab.txt` que será criado também em `home`. Note o uso do `>>`, que significa que sempre será inserido o conteúdo da listagem sempre após a última linha do arquivo `crontab.txt`. Mas e esse monte de `*` ?

Cada `*` representa uma configuração no `crontab`, que indica:

- `minuto`: vai de 0 a 59;
- `hora`: vai de 0 a 23;
- `dia`: vai de 1 a 31;
- `mês`: vai de 1 a 12;
- `dia da semana`: vai de 0 a 7.

Além dessas configurações, ainda podemos definir qual usuário irá executar a tarefa agendada. Em seguida, definimos o comando ou instrução para execução. Ainda é possível utilizar os operadores `, - /` , sendo que a `(,)` serve para especificar uma lista de valores, o `-` serve para especificar intervalos de valores, e a `/` serve para pular valores.

Se você implementou o agendamento de exemplo, já deve ter se passado mais de 1 minuto, então verifique na listagem da pasta `home` do seu usuário se o arquivo `crontab.txt` foi criado e se existe informação nele:

```
daniel@casadocodigo:~$ ls cron*
crontab.txt
daniel@casadocodigo:~$ tail -n 3 crontab.txt
relatorio
verificador.sh
vim_basico.txt
daniel@casadocodigo:~$
```

Se o arquivo existe e contém as informações da listagem da pasta `home`, nosso agendamento funcionou perfeitamente!

Agora que entendemos como funciona o agendamento de tarefas com o `crontab`, podemos prosseguir e construir um script de backup e implementar nossa solução automatizada de backup agendado.

10.6 Um simples script de backup

Vamos implementar agora o script para realizar o backup do diretório `home` do nosso usuário. Para isso, usaremos o comando `tar` e o comando `date`. O `tar` vai gerar uma cópia do `home` compactada, o `date` irá incrementar as cópias adicionando a informação do dia, mês, ano, hora e minuto em que o backup foi realizado.

Primeiro vamos criar uma pasta chamada `backup` no diretório raiz `(/)` do nosso sistema:

```
daniel@casadocodigo:~$ sudo mkdir /backup
```

Crie um arquivo chamado `backup.sh`. Use um editor de sua escolha e implemente o seguinte script:

```
#!/bin/bash
# Script de backup /home/daniel

sudo tar czf /backup/$(date +%d%m%Y-%H%M)-home.tar.gz -C /home/daniel .
```

Estamos fazendo uma cópia da pasta `home` do usuário `daniel` e compactando no formato `tar.gz`. Ao criar a cópia, estamos adicionando a data atual e a hora no momento do backup. Para ver funcionando, altere a permissão do script para que ele possa ser executado, em seguida rode o programa:

```
daniel@casadocodigo:~$ chmod 777 backup.sh
daniel@casadocodigo:~$ ./backup.sh
daniel@casadocodigo:~$
```

Mas o que aconteceu? O backup foi feito? Olhe o diretório `/backup`:

```
daniel@casadocodigo:~$ ls -l /backup
total 12
-rw-r--r-- 1 root root 9935 Oct 13 23:32 13102013-2332-home.tar.gz
daniel@casadocodigo:~$
```

O nosso backup foi realizado com sucesso! Note que o nome do arquivo gerado contém a data e hora da criação `13102013-2332-home.tar.gz` – para ler melhor, imagine o seguinte formato: `13/10/2013-23:32` . Faça um teste com o comando `date` e veja sua saída padrão; em seguida, rode o mesmo comando com a formatação que passamos no script de backup:

```
daniel@casadocodigo:~$ date
Sun Oct 13 23:38:09 UTC 2013
daniel@casadocodigo:~$ date +%d%m%Y-%H%M
13102013-2338
daniel@casadocodigo:~$
```

Agora que temos o nosso script de backup, podemos criar uma tarefa `cron` para que ele seja executado a cada 5 minutos, por exemplo. Vamos criar o agendamento na conta do nosso usuário, para isso usaremos o `crontab`:

```
daniel@casadocodigo:~$ crontab -e -u daniel
```

Edite o arquivo na última linha e altere a tarefa que criamos anteriormente para:

```
*/5 * * * * /home/daniel/backup.sh
```

Estamos informando ao `crontab` para executar o nosso script `backup.sh` a cada 5 minutos.

Se você quiser praticar mais, você pode criar um script que faz o backup de um determinado banco de dados do MySQL. Para isso, utilize o comando `mysqldump`, que é instalado junto com o pacote do MySQL.

10.7 Personalizando o seu shell: PS1, PATH e outros

O `shell` pode ser totalmente personalizado, podemos alterar o prompt de comando, esquema de cores, automatização de processos e muito mais. Veremos agora o básico para customizá-lo.

Um pouco sobre variáveis

Sendo Shell Script uma linguagem de programação para o ambiente `shell`, temos recursos da linguagem disponíveis por padrão para personalizar. Lembra das variáveis em Shell Script? Elas podem ser criadas e acessadas diretamente no console:

```
daniel@casadocodigo:~$ NOME="Daniel Romero"
daniel@casadocodigo:~$ echo $NOME
Daniel Romero
daniel@casadocodigo:~$
```

Como você viu anteriormente, em Shell Script, ao criar uma variável é possível ter acesso ao seu conteúdo armazenado fazendo uso do operador (`$`) – como acabamos de ver, isso funciona diretamente no `shell`.

É importante saber que existem variáveis locais e globais, sendo que as variáveis locais são restritas a um escopo definido e as globais ficam disponíveis para todo o sistema. Estas últimas são bastante conhecidas por variáveis de ambiente. A diferença entre elas está na criação:

```
VARIAVEL_LOCAL="valor"
export VARIAVEL_GLOBAL="valor"
```

Para definir uma variável local, só precisamos atribuir um valor ao nome da variável. Para definir uma variável de ambiente, é necessário fazer uso do comando `export` antes de sua definição.

Este conceito de variáveis é importante pois é através delas que definimos a forma como o `shell` e outros aplicativos irão se comportar.

Um exemplo de variável global é a `PS1`, que é responsável por guardar as configurações do prompt de comandos. Essas configurações apresentam os dados do computador, nome de usuário, diretórios. Por exemplo, quando estamos no `shell` temos o seguinte: `daniel@casadocodigo:~$`. Esta informação é criada e mantida na variável `PS1`, vamos ver o que está armazenado nela:

```
daniel@casadocodigo:~$ echo $PS1
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$
daniel@casadocodigo:~$
```

Esses códigos armazenados em `PS1` é que definem a forma do prompt. Podemos alterar isso a nosso gosto, mas antes vamos conhecer mais detalhes sobre o `shell` para em seguida modificá-lo.

Uma outra variável de sistema muito importante é o `PATH`, que armazena os caminhos absolutos de arquivos binários (executáveis) de aplicativos. Eles podem ser chamados sem a necessidade de informarmos o caminho completo.

```
daniel@casadocodigo:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
daniel@casadocodigo:~$
```

Para melhor entender, busque o caminho absoluto do `vim`. Para isso, utilize o comando `whereis`:

```
daniel@casadocodigo:~$ whereis vim
vim: /usr/bin/vim /usr/bin/vim.basic /usr/bin/vim.tiny
     /etc/vim /usr/bin/X11/vim /usr/bin/X11/vim.basic
```

```
    /usr/bin/X11/vim.tiny /usr/share/vim
    /usr/share/man/man1/vim.1.gz
daniel@casadocodigo:~$
```

Como o `$PATH` armazena o trajeto `/usr/bin`, não precisamos informar o caminho inteiro ao executar o aplicativo `vim`, que seria `/usr/bin/vim`. Isto serve para praticamente quase todos os aplicativos. Podem aparecer casos em que seja necessário alterar o valor da variável `PATH`, o que pode ser feito em `/etc/profile` ou de maneira mais organizada separando por scripts em `/etc/profile.d`.

10.8 Alias

Outro recurso que usaremos bastante é o `alias`, que cria um apelido para uma determinada instrução. Sua sintaxe é parecida com a definição de uma variável.

Vamos ver a lista de `alias` que existe definida no sistema:

```
daniel@casadocodigo:~$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
daniel@casadocodigo:~$
```

Perceba que a declaração é a mesma de uma variável, sendo que o comando `alias` é utilizado antes. Este recurso é bastante útil para não termos que repetir comandos grandes que recebem parâmetros. Por exemplo, tome o comando `ls --color=auto` que ativa um esquema de cores para exibir o conteúdo listado de forma mais agradável. Note que o `alias` para o comando `ls` recebe como argumento o comando `ls --color=auto`, assim, sempre que fizermos uso do `ls`, ele já vai exibir a saída em cores.

Podemos criar o nosso próprio `alias`, por exemplo, para o comando `clear`. Em vez de digitarmos o `clear` para limpar o buffer do terminal, podemos ter um apelido chamado `cl`. Para isso, faça:

```
daniel@casadocodigo:~$ alias cl='clear'
daniel@casadocodigo:~$
```

Agora podemos executar o comando `clear` chamando pelo seu apelido `cl`. Neste exemplo, o `cl` só vai existir enquanto esta sessão do `shell` estiver aberta. Veremos mais adiante como tornar esse apelido fixo assim como os demais existentes.

10.9 Arquivos de configuração

Quando nos logamos no sistema e abrimos um `shell`, esse shell é de login interativo e executa várias instruções de configurações para, por exemplo, customizar o `bash` ou disponibilizar um novo `alias`.

Por padrão, ao logar no `shell`, ele vai ler o arquivo `/etc/profile` e executá-lo, aplicando as configurações para todos os usuários do sistema. Na sequência, após ler o `/etc/profile`, irá buscar por um dos seguintes arquivos, na pasta `home` do usuário:

- `~/.profile`
- `~/.bash_profile`
- `~/.login`
- `~/.bashrc`

Caso um script seja executado e na sua programação ele faça uso do `bash`, ele irá apenas executar o `~/.bashrc`, já que durante essa chamada não se trata de uma `shell` efetuando uma operação de login.

Para entender melhor, vamos criar um `alias` e defini-lo para ser carregado logo que chamamos um `shell`. No exemplo, vamos setar um novo `alias` apenas para o usuário `daniel`. Para isso, precisamos editar o arquivo `.bashrc` que existe na pasta `home` do nosso usuário:

```
daniel@casadocodigo:~$ vim .bashrc
```

Observe as várias configurações neste arquivo, vamos editá-lo seguindo a sua organização. Existe uma marcação específica para inserir aliases, vamos inserir o nosso `alias` nesse grupo:

```
# some more ls aliases
alias ll='ls -alF'
```

```
alias la='ls -A'
alias l='ls -CF'
alias cl='clear' # <--- Nosso novo alias
```

Salve o arquivo e saia do editor. Agora teste o seu novo alias e execute a sua chamada:

```
daniel@casadocodigo:~$ cl
The program 'cl' is currently not installed.
You can install it by typing:
sudo apt-get install cl-launch
daniel@casadocodigo:~$
```

O que aconteceu? Por que falhou? Nós criamos o `alias` no arquivo certo porém ainda não informamos ao `shell` que ele deve ler o script `.bashrc` e aplicar a nova configuração. Podemos resolver isso de 2 formas, a primeira é bem simples: encerre o terminal e abra um novo. A segunda maneira é mais interessante, não precisamos encerrar o nosso terminal: para aplicar a nova configuração vamos utilizar o comando `source` que irá interpretar o script e aplicar a nova configuração na sessão `shell` que já está aberta. Para isso, faça:

```
daniel@casadocodigo:~$ source .bashrc
daniel@casadocodigo:~$
```

Em seguida, rode o comando `cl` novamente e veja que tudo funciona.

Além do `.bashrc` no diretório `home` do nosso usuário ainda podemos encontrar mais 2 arquivos de configuração:

```
daniel@casadocodigo:~$ ls .b*
.bash_history  .bash_logout  .bashrc
daniel@casadocodigo:~$
```

O `.bash_history` armazena tudo o que foi digitado no `shell`. Esse arquivo é lido pelo comando `history` como já vimos anteriormente. Já o `.bash_logout` executa instruções logo que saímos do `shell`, por exemplo, ao rodar o comando `exit` e efetuar o logout, esse script será executado.

Capítulo 11

Compilando arquivos fonte

Compilar ou recompilar um software a partir do código fonte é necessário sempre que precisamos alterar alguma configuração específica que não veio disponível na sua versão pré-compilada, por exemplo, quando instalamos via `apt`. Além disso, a compilação é bastante utilizada para aplicar atualizações de segurança e habilitar novas funcionalidades no software.

Neste capítulo, veremos o básico de compilação. Vamos instalar o pacote da linguagem de programação `ruby` fazendo algumas alterações durante a compilação, e entender como funciona esse processo. Algumas vezes ele pode ser mais complicado, às vezes menos. Depende muito de como os desenvolvedores prepararam o pacote.

11.1 A configuração

Sempre que vamos instalar um software a partir do seu código fonte, veremos um arquivo chamado `configure`. Ele é, na verdade, um aplicativo que, ao ser execu-

tado, verifica no sistema se existem todas as dependências necessárias para compilar o software em questão.

O aplicativo `configure` disponibiliza a opção `help`, que exibe uma lista de opções que pode ser utilizada no processo de compilação. Essas opções incluem as funcionalidades do software que podem ser habilitadas ou não.

Após a execução do `configure`, se for concluído com sucesso, ele irá gerar o arquivo `Makefile` que veremos a seguir.

O arquivo `Makefile` é responsável por automatizar o processo de compilação, além de fazer a verificação para ter certeza de que nada faltou e por fim à instalação do software.

11.2 Compilando na prática

Para começar, vamos fazer o download do código fonte do `ruby`. Para isso, usaremos um comando que ainda não vimos, o `wget`.

O `wget` é um aplicativo `shell` para executar tarefas de download muito poderoso e tem várias possibilidades de uso. Faça o download do `ruby` executando o comando:

```
daniel@casadocodigo:~$ wget -c \
> http://cache.ruby-lang.org/pub/ruby/2.0/ruby-2.0.0-p247.tar.gz
```

A opção `-c` do comando `wget` é para continuar o download de onde parou caso ele seja interrompido. O uso da `\` foi apenas para formatar o comando no terminal.

Após concluir o download, descompacte o arquivo para o diretório `/usr/local`. É nele que iremos compilar o `ruby`:

```
daniel@casadocodigo:~$ sudo tar zxvf ruby-2.0.0-p247.tar.gz -C
/usr/local
```

Lembre-se de que a opção `-C` do comando `tar` serve para indicarmos o diretório destino ao descompactar o arquivo. Após descompactar, vamos ao diretório onde está o código fonte do `ruby`:

```
daniel@casadocodigo:~$ cd /usr/local/ruby-2.0.0-p247/
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ ls
addr2line.c       inits.c           regexec.c
addr2line.h       insns.def         regint.h
array.c           insns.inc         regparse.c
```

```
bcc32           insns_info.inc     regparse.h
benchmark       internal.h         regsyntax.c
bignum.c        io.c               revision.h
bin             iseq.c             ruby_atomic.h
bootstraptest   iseq.h             ruby.c
BSDL            KNOWNBUGS.rb       safe.c
ChangeLog       known_errors.inc   sample
class.c         LEGAL              signal.c

...
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$
```

Podemos ver que no diretório do código fonte existe bastante coisa! A primeira coisa a fazer nesse momento é ler os arquivos `README` e `INSTALL`. Dependendo do desenvolvedor do software, ele pode disponibilizar apenas um dos arquivos – no nosso caso, existe o `README` com informações sobre o software e procedimentos de instalação.

Prosseguindo, vamos ao aplicativo `configure` ver as opções de compilação:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ ./configure --help
```

A quantidade de opções para o processo de compilação realmente é muito grande. Escolheremos apenas algumas: vamos desabilitar a instalação do suporte da linguagem `ruby` para comunicação com a linguagem `C` e vamos desabilitar também a instalação da documentação `rdoc` da linguagem `ruby`:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ sudo ./configure \
> --disable-install-capi --disable-install-rdoc
```

Mais uma vez estamos usando \ apenas para formatação no `shell`. O comando pode ser inserido inteiro em uma única linha. As duas opções escolhidas para desabilitar no exemplo são apenas para ilustrar o uso do processo de compilação.

Após rodar o aplicativo `configure`, receberemos a seguinte mensagem no final: `config.status: creating Makefile`, indicando que o arquivo `Makefile` foi criado. Agora podemos ir para a próxima etapa e compilar o software:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ sudo make
```

Para compilar, usamos o comando `make`, que irá basicamente ler o arquivo `Makefile` com as instruções de compilação e executar cada etapa. O processo de

compilação pode demorar um pouco. Neste exemplo a compilação pode levar entre 1 e 3 minutos, dependendo do seu hardware.

Após concluir, se não houve nenhuma mensagem de erro gerada pelo comando `make`, podemos finalizar executando a instalação:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ sudo make install
```

A função do `make install` é essencialmente mover alguns arquivos que foram compilados para os seus devidos lugares e, assim, efetivar a instalação. Durante todas essas etapas foram gerados alguns arquivos binários que servem de auxílio durante a compilação. Agora esses arquivos podem ser removidos, e para isso use a opção `clean` do comando `make`:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ sudo make clean
```

Retorne para o diretório `home` do seu usuário e teste para ver se o aplicativo foi instalado com sucesso:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ cd
daniel@casadocodigo:~$ ruby -v
ruby 2.0.0p247 (2013-06-27 revision 41674) [i686-linux]
daniel@casadocodigo:~$
```

Executando `ruby -v`, se você recebeu a mensagem informando a versão do `ruby` que instalamos, significa que tudo foi executado corretamente.

Como o `ruby` poderia ser instalado rapidamente via `apt`, podemos remover a nossa instalação executando a opção `uninstall` do comando `make`. Para isso vamos voltar para o diretório onde se encontra o código fonte da instalação:

```
daniel@casadocodigo:~$ cd -
/usr/local/ruby-2.0.0-p247
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ sudo make uninstall
make: *** No rule to make target `uninstall`.  Stop.
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$
```

Mas o que aconteceu? Não foi possível desinstalar pois o criador do software não disponibilizou a opção `uninstall` para o `make`. Mas ainda temos duas formas para desinstalar.

A primeira forma é mais trabalhosa, teríamos que localizar os diretórios onde o `ruby` foi instalado no sistema. Poderíamos usar o comando `whereis`:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ whereis ruby
ruby: /usr/local/bin/ruby /usr/local/lib/ruby
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$
```

E sabendo agora onde estão os arquivos da instalação do `ruby` no sistema, teríamos que remover manualmente, mas isso é muito trabalho. Podemos utilizar o `shell` com mais sabedoria e remover tudo de uma vez.

Ao realizar a instalação, o `make` gerou um arquivo oculto chamado `.installed.list`, no qual está exatamente a estrutura dos diretórios e arquivos que foram instalados durante o processo de compilação:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ cat .installed.list
/usr/local/bin/
/usr/local/bin/ruby
/usr/local/lib/
/usr/local/lib/libruby-static.a
/usr/local/lib/ruby/2.0.0/i686-linux/
/usr/local/lib/ruby/2.0.0/i686-linux/rbconfig.rb
/usr/local/lib/pkgconfig/
/usr/local/lib/pkgconfig/ruby-2.0.pc
/usr/local/bin/testrb
/usr/local/bin/ri
...
```

Agora que temos a lista dos arquivos com os respectivos caminhos absolutos podemos montar uma instrução de uma única linha para remover todos do sistema – para isso usaremos o `xargs`. Basicamente, o que o `xargs` faz é combinar o parâmetro inicial com os argumentos recebidos da entrada padrão, e dessa forma ele executa o comando ordenado uma ou mais vezes.

Para remover os arquivos da instalação que estão listados em `.installed.list` usaremos o `cat` para ler, o `rm` para remover e o `xargs` para fazê-los se comunicarem:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ cat .installed.list |
xargs sudo rm
```

Não se preocupe com as mensagens de alerta desta instrução pois estamos removendo apenas os arquivos e não os diretórios. Teste agora o comando `ruby -v` e veja que não está mais instalado no sistema:

```
daniel@casadocodigo:/usr/local/ruby-2.0.0-p247$ cd
daniel@casadocodigo:~$ ruby -v
bash: /usr/local/bin/ruby: No such file or directory
daniel@casadocodigo:~$
```

Lembre-se sempre de ler o `README` ou `INSTALL` dos programas que for instalar pelo código fonte!

Já está preparado para mais pacotes? Procure e baixe o pacote `nmap`, que serve para escanear a rede para segurança, e tente compilá-lo e instalá-lo, seguindo passos análogos aos já vistos aqui.

Capítulo 12

O que estudar além?

Agora que você já se familiarizou com o Linux, está na hora de partir em busca de mais conhecimentos e avançar nos estudos. Selecionei algumas dicas interessantes para que você prossiga com o aprendizado!

12.1 SSH – Secure Shell

`SSH` (Secure Shell) é um protocolo que estabelece um canal de comunicação seguro entre dois computadores por meio de criptografia e codificação de mensagens de autenticação. O `SSH` é bastante utilizado quando queremos estabelecer uma conexão com máquinas remotas e executar comandos em um servidor remoto, por exemplo.

A forma mais comum de utilização é: `ssh username@servidor_ip`.

```
$ ssh daniel@192.168.25.8

The authenticity of host '192.168.25.8 (192.168.25.8)'
can't be established.
```

```
ECDSA key fingerprint is
32:53:5d:95:d9:2b:c0:92:ab:1d:a4:87:95:a6:5a:e2.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.25.8' (ECDSA) to the list of known
hosts.
daniel@192.168.25.8's password:
Welcome to Ubuntu 12.04.3 LTS (GNU/Linux 3.2.0-23-generic-pae i686)

daniel@casadocodigo:~$
```

No exemplo, estabeleci uma conexão remota em uma maquina virtual passando o nome do usuário e ip como argumentos do comando `ssh`. Durante a conexão, o `ssh` solicita a senha do usuário para completar o acesso remoto.

Imagine a seguinte situação: você está em seu computador e precisa acessar outro computador utilizando `ssh` via internet. Durante o processo de login você informou a sua senha, mas no meio do caminho tinha alguém observando os seus passos:

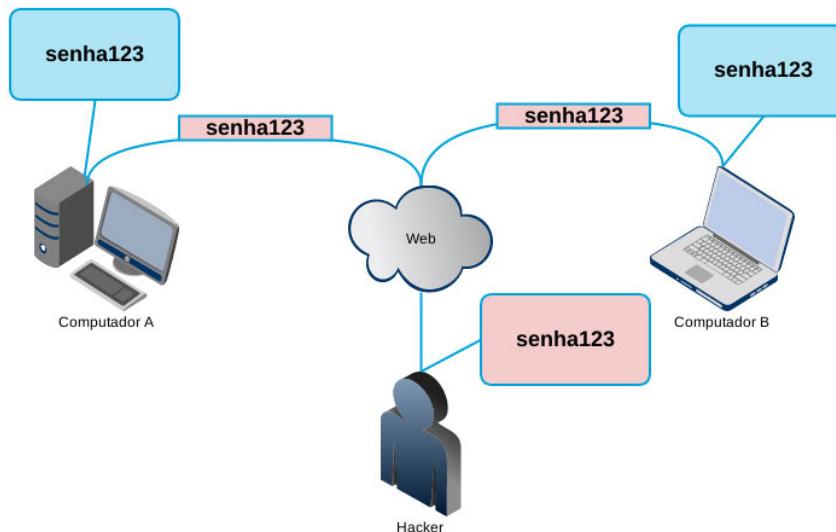



Figura 12.1: Acesso ssh inseguro

Na imagem anterior é possível notar que os dados estão desprotegidos e que qualquer pessoa consegue capturar e entender as informações que trafegam entre o `computador A` e o `computador B`, pois as informações não estão criptografadas.

Utilizando criptografia, os dados que saem do seu computador seriam transformados em algo totalmente incompreensível para qualquer pessoa que não fosse o destinatário.

Podemos notar que ainda é possível interceptar as informações que trafegam entre o `computador A` e o `computador B` mas é muito difícil entender o conteúdo:

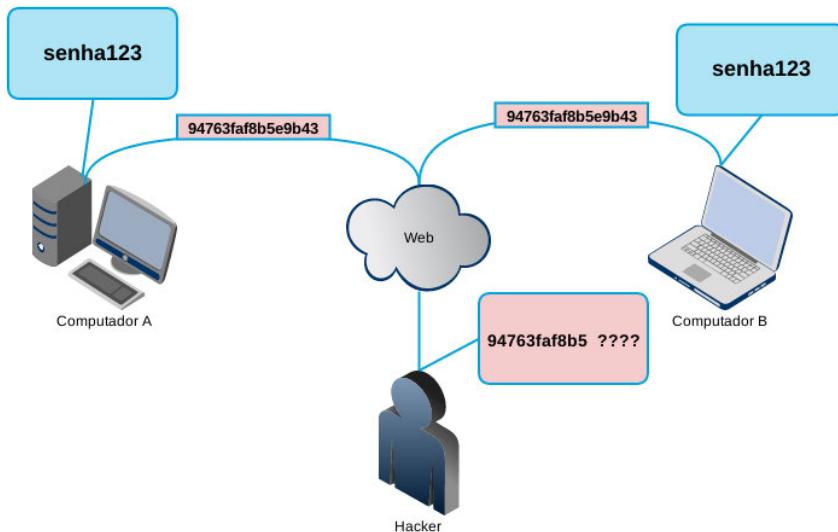



Figura 12.2: Acesso ssh seguro

Note que agora o nosso hacker não conseguiu decifrar as informações trocadas entre os computadores `A` e `B`, graças à criptografia.

O `SSH` utiliza um sistema de criptografia que requer 2 chaves, uma chave é conhecida por `chave pública` que pode ser usada pelo remetente para criptografar os dados antes de enviá-los e a outra é conhecida por `chave privada` que é usada para decodificar a mensagem original.

Para o nosso exemplo, vamos gerar um par de chaves e garantir o nosso acesso seguro via `SSH`. No linux, o `OpenSSH` trabalha com dois tipos de chaves: `DSA (Digital Signature Algorithm)` e `RSA (Rivest, Shamir e Adleman)`, as chaves do tipo `DSA` são padronizadas pelo governo dos EUA e o `OpenSSH` só cria chaves DSA com no máximo 1024 bits. Isso pode ser um problema pois seria possível decodificar uma chave DSA a curto prazo, já as chaves RSA

o `OpenSSH` pode gerar chaves com até 4096 bits ou até mais e isso as torna inquebráveis na atualidade.

Utilizaremos o padrão do `OpenSSH` que é de chaves `RSA` de 2048 bits. Para gerar as chaves usaremos o aplicativo `ssh-keygen`:

```
$ ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/home/daniel/.ssh/id_rsa):
```

O `ssh-keygen` vai perguntar onde desejamos salvar as chaves que serão criadas. Por padrão é salvo na pasta `.ssh` na `home` do nosso usuário, então nesta etapa, apenas tecle `Enter`.

Em seguida será solicitada uma senha e a confirmação de senha para a geração das chaves. Por comodidade vamos deixar em branco e apenas teclar `Enter`:

```
$ ssh-keygen

Generating public/private rsa key pair.
Enter file in which to save the key (/home/daniel/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/daniel/.ssh/id_rsa.
Your public key has been saved in /home/daniel/.ssh/id_rsa.pub.
The key fingerprint is:
ce:71:1f:c2:54:b4:e6:53:bf:59:72:64:d5:be:46:a7 daniel@localhost
The key's randomart image is:
+--[ RSA 2048]----+
|           .o   o|
|           . .  o|
|          . o ..o|
|         o o . =o|
|        S + + o.*|
|       o o o oE=+|
|        o   . .o |
|                 |
|                 |
+-----------------+
```

Pronto, o nosso par de chaves foi gerado. Para verificar basta listar as chaves:

```
$ ls ~/.ssh
id_rsa id_rsa.pub
```

Ao listar os arquivos no diretório `.ssh` na pasta home do nosso usuário temos o par de chaves gerado onde `id_rsa` é a chave privada e `id_rsa.pub` nossa chave pública.

Agora que já possuímos nosso par de chaves o que precisamos fazer é levar a nossa chave pública para o destinatário e dizer para ele que iremos estabelecer comunicação com o uso de chaves. Para isso utilizaremos o `scp (secure copy)` um aplicativo de cópia segura de arquivos que utiliza o ssh como canal de envio:

```
$ scp ~/.ssh/id_rsa.pub daniel@192.168.25.8:~/chave_publica
daniel@192.168.25.8's password:
```

O que fizemos foi informar ao `scp` o arquivo a ser enviado e o destino, sendo que o esse é a mesma instrução que utilizamos para uma conexão ssh.

Após enviar nossa chave pública, precisamos entrar e configurar no servidor para que o mesmo passe a utilizar a comunicação por chaves no `ssh`:

```
$ ssh daniel@192.168.25.8
daniel@192.168.25.8's password:
Welcome to Ubuntu 12.04.3 LTS (GNU/Linux 3.2.0-23-generic-pae i686)

daniel@casadocodigo:~$
```

Agora que entramos no servidor novamente, só precisamos configurar para que ele leia nossa chave pública:

```
daniel@casadocodigo:~$ cat chave-publica >> ~/.ssh/authorized_keys
```

O arquivo `authorized_keys` é responsável por autorizar quem pode ou não manter acesso ao servidor via `ssh`. Agora basta verificar se o `OpenSSH` está configurado para aceitar login utilizando chaves:

```
daniel@casadocodigo:~$ sudo grep 'PubkeyAuthentication'
                                        /etc/ssh/sshd_config
PubkeyAuthentication yes
daniel@casadocodigo:~$ exit
```

No meu caso a opção está ativa. Se a saída do comando retornar `no` em vez de `yes`, abra o arquivo `/etc/ssh/sshd_config` e altere para `yes`.

Agora é possível manter acesso seguro via ssh sem a necessidade de informar senha:

```
$ ssh daniel@192.168.25.8
daniel@casadocodigo:~$
```

Por medidas de segurança, nunca revele o conteúdo da sua chave privada.

Para mais detalhes, é interessante olhar a documentação oficial da distribuição Linux que estamos usando, no caso o Ubuntu: https://help.ubuntu.com/community/SSH

12.2 Proteção por firewall

O `firewall` tem por objetivo aplicar uma política de segurança por meio de regras que controlam o fluxo de entrada e saída de dados em uma rede e proteger os computadores.

No Linux, o aplicativo de `firewall` mais conhecido é o `iptables` que é bastante poderoso e complexo. No Ubuntu, existe uma ferramenta que ajuda na configuração do `firewall` reduzindo a complexidade do `iptables`: UFW (Uncomplicated Firewall) http://wiki.ubuntu-br.org/UFW

Em versões futuras no Kernel Linux, o `iptables` será substituído por `nftables` http://netfilter.org/projects/nftables/ para reduzir a complexidade da escrita de regras que atualmente há no `iptables`.

12.3 Upstart e Monit

Lembra dos scripts de inicialização de serviços? O `upstart` é uma solução ótima disponível também no Ubuntu. Com ele é possível criar scripts de inicialização e controle de serviços de forma fácil e menos dolorosa: http://upstart.ubuntu.com/index.html.

Já o `monit` é um aplicativo para gerenciamento e monitoramento de processos, aplicativos, arquivos, diretórios etc. Ele pode executar ações planejadas e é bastante indicado para monitorar servidores. Além disso, ele pode trabalhar de forma parecida com o `cron`: http://mmonit.com/monit/.

12.4 Documentações em português

Se você não se sentir confortável com material em inglês, existe uma vasta documentação sobre Linux em português.

O guia foca http://www.guiafoca.org/ é uma documentação bastante completa separada em níveis iniciante, intermediário e avançado.

12.5 Tirar dúvidas

Se você tiver alguma dúvida sobre este livro, junte-se à lista em:

http://lista.infoslack.com

E envie sua pergunta! Eu e os participantes do grupo tentaremos ajudar.

MUITO OBRIGADO!

Índice Remissivo